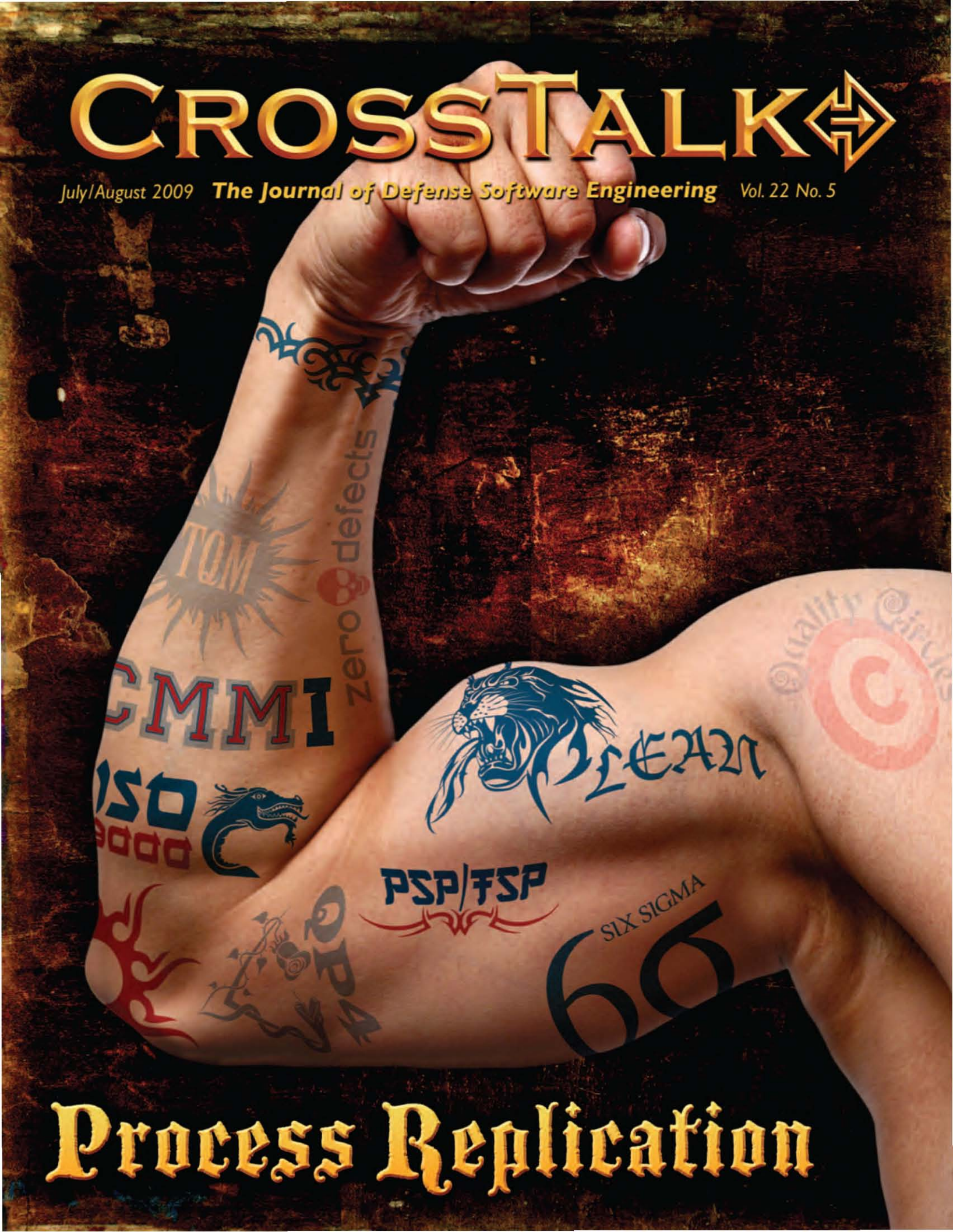

CROSSTALK
July/August 2009 The Journal of Defense Software Engineering Vol. 22 No. 5
TQM
zero defects
CMMI
ISO
9000
LEAN
PSP/TSP
SIX SIGMA
6σ
Process Replication

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
AUG 2009		**00-07-2009 to 00-08-2009**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Crosstalk, The Journal of Defense Software Engineering. Volume 22, Number 5	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **CrossTalk / 517 SMXS/MXDEA,6022 Fir AVE,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

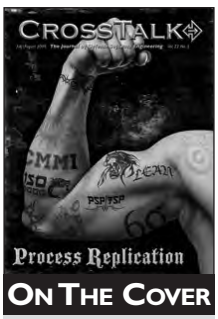
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**36**	

Process Replication

Software Engineering Technology

Open Forum



On The Cover
Cover Design by Kent Bingham

Departments

CrossTalk

OSD (AT&L) *Kristin Baldwin*
NAVAIR *Joan Johnson*
309 SMXG *Karl Rogers*
DHS *Joe Jarzombek*

Managing Director *Brent Baxter*
Publisher *Kasey Thompson*
Managing Editor *Drew Brown*
Associate Editor *Chelene Fortier-Lozancich*
Publishing Coordinator *Nicole Kentta*
Phone (801) 775-5555
E-mail stsc.customerservice@hill.af.mil
CrossTalk Online www.stsc.hill.af.mil/crosstalk

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the Office of the Secretary of Defense (OSD) Acquisition, Technology and Logistics (AT&L); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CrossTalk, providing both editorial oversight and technical review of the journal. CrossTalk's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 33.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CrossTalk does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the author and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CrossTalk.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.web master@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Process Replication and Model-Based Process Improvement

NAVAIR

CROSSTALK would like to thank NAVAIR for sponsoring this issue.

You may be asking, "What exactly is process replication?" It is the ability to take an effective, well-defined process framework from one environment and apply it to another that provides similar products and services. Process replication enables engineers and their teams to hit the ground running.

Processes are what we develop, document, and then use to train people to do good work in a repeatable way. From this, we expect those we've trained to be enabled to produce high quality products on cost and on schedule. To maximize success, these processes are based upon best practices found in capability maturity models. The processes are the "how" and the models are the "what." An important thing to remember is that they are complementary.

At NAVAIR, the Team Software Process (TSP) has been an extremely effective way of carrying out process replication. It is a complete, ready-to-go process with training for multiple levels of a project and its organization, process scripts for project planning and operations, planned coaching to help teams maintain direction, and a choice of tools that automate its use. All of this creates a team environment that both supports disciplined individual work and builds and maintains a self-directed team. The TSP has continually proven its ability to replicate self-directed teams that address critical business areas, including cost and schedule management, effective quality management, and cycle-time reduction.

Furthermore, the fundamental principles of this team process can be replicated with many other engineering and technical teams to plan and operate their work of delivering products and providing services: for instance, in requirements elicitation and definition, design, implementation, test, maintenance—and even in process improvement itself.

The team process builds on and enables the replication of personal process. These personal processes show individuals how to measure their work and use that data to improve their performance. This, in turn, guides individual feedback to the team process, accelerating teamwork and creating an environment where individuals come together as a self-directed team that can use data to both know where they stand and more effectively plan for the future.

Experiences in organizations both within NAVAIR and beyond have shown that success in organizations pursuing model-based process improvement were accelerated when done in conjunction with their projects using these replicated processes.

Four articles in this issue address various experiences in the area of processes ready for replication. In *Lean Enablers for Systems Engineering* (SE), Dr. Bohdan W. Oppenheim presents a collection of "dos" and "don'ts" of SE with ready-to-go wisdom on how to prepare for, plan, execute, and practice SE and related enterprise management using Lean thinking. In *A Framework for Systems Engineering Development of Complex Systems*, Dr. Karl L. Brunson, Dr. Jeffrey Beach, Dr. Thomas A. Mazzuchi, and Dr. Shahram Sarkani present an SE process—called the Incremental Commitment Model—where schedule tasks are evaluated against technical and manufacturing risk drivers. In *Why Software Requirements Traceability Remains a Challenge*, Andrew Kannenberg and Dr. Hossein Saiedian provide a brief introduction to software requirements traceability and investigate why many challenges exist in traceability practices today. Dr. Christof Ebert's article, *Software Project and Process Measurement*, briefly introduces us to software measurement and provides practical guidance for project and process measurement.

Two supporting articles will also help organizations with process improvement and SE: Larry Pizette, Salim Semy, Geoffrey Raines, and Steve Foote's *A Perspective on Emerging Industry SOA Best Practices* will help organizations improve business processes while David P. Quinn sees *Resistance as a Learning Opportunity* for process improvement in software and systems engineering.

I invite you to consider these experiences and approaches as additional ways to make process improvement happen for you and in your organization. Best practice models and process replication together do make our engineering products better, faster, and cheaper.

Jeff Schwalb
Naval Air Systems Command Co-Sponsor

Lean Enablers for Systems Engineering

Dr. Bohdan W. Oppenheim
Loyola Marymount University

A new product named Lean Enablers for Systems Engineering (LEfSE) is described. It is a collection of 194 practices and recommendations formulated as "dos" and "don'ts" of SE, and containing collective wisdom on how to prepare for, plan, execute, and practice SE and related enterprise management using Lean Thinking. The enablers are focused on mission assurance and the satisfaction of stakeholders achieved with minimum waste. The product has been developed by experts from the Lean Systems Engineering (LSE)[1] Working Group (WG) of the International Council on Systems Engineering (INCOSE). LEfSE are organized into six well-known Lean Principles called Value, Value Stream, Flow, Pull, Perfection, and Respect for People. The LEfSE are not intended to become a mandatory practice. Instead, they should be used as a checklist of good practices.

Systems engineering is regarded as an established sound practice but not always delivered effectively. Sixty-two recent successful space launches indicate that mission assurance can be practiced well. At the same time, recent U.S. Government Accountability Office (GAO) and NASA studies of space systems [1, 2, 3, 4] document notorious major budget and schedule overruns, some exceeding 100 percent. Most programs are burdened with waste, poor coordination, unstable requirements, quality problems, and management frustrations. Recent studies by the MIT-based Lean Advancement Initiative (LAI) researchers [5, 6, 7, 8] have identified a mind-boggling amount of waste in government programs, reaching 70 percent of charged time. This waste represents a vast productivity reserve in programs and major opportunities to improve program efficiency.

The new field of LSE is the application of Lean Thinking to SE and to the related aspects of enterprise management. SE is focused on the flawless performance of complex technical systems. Lean Thinking is the holistic management paradigm credited for the extraordinary rise of Toyota to the most profitable and the largest auto company in the world [9]. Toyota is well-known for practicing excellent Product Development and SE (what Toyota refers to as simultaneous engineering). For example, the Prius car design was completed in nine months from the end of styling, a performance level unmatched by any competitor [10]. Lean Thinking has been successfully applied in defense industry and in the U.S. military itself, (e.g., [5], and the Air Force Lean initiative named AFSO-21). It has become an established paradigm in manufacturing, aircraft depots, administration, supply chain management, health, and Product Development, including engineering.

LSE is the area of synergy of Lean and SE with the goal to deliver the best life-cycle value for technically complex systems with minimal waste. LSE does not mean *less SE*. It means more and better SE with higher responsibility, authority, and accountability (RAA), leading to better and waste-free workflow and mission assurance. Under the LSE philosophy, mission assurance is non-negotiable, and any task which is legitimately required for success must be included, but it should be well-planned and executed with minimal waste.

Fundamentals of Lean Thinking

Three concepts are fundamental to the understanding of Lean Thinking: value, waste, and the process of creating value without waste (also known as Lean Principles).

Value

The value proposition in engineering programs is often a multi-year complex and expensive acquisition process, involving thousands of stakeholders and resulting in hundreds or even thousands of requirements, which, notoriously, are rarely stable (even at the Request for Proposal phase). In Lean SE, Value is defined simply as mission assurance (the delivery of a flawless complex system, with flawless technical performance during the product or mission life cycle), satisfying the customer and all other stakeholders, which implies completion with minimal waste, minimal cost, and the shortest possible schedule.

Waste in Product Development

LAI classifies waste into seven categories: 1) Overproduction; 2) Transportation; 3) Waiting; 4) Over-processing; 5) Inventory; 6) Unnecessary movement; and 7) Defects. These wastes, in the SE context, are elaborated on in [11].

Lean Principles

Womack [9] captured the process of creating value without waste into six Lean Principles[2]. The Principles are abbreviated as Value, Value Stream, Flow, Pull, Perfection, and People, defined as follows:

1. **The customer (either external or internal) defines value.** The value proposition must be captured with crystal clarity early in the program.
2. **Map the value stream.** Prepare for and plan all end-to-end linked actions and processes necessary to realize value, streamlined, after eliminating waste.
3. **Make value flow continuously.** This should happen without stopping, rework, or backflow (legitimate optimized iterations are okay).
4. **Let (internal or external) customers pull value.** The customer's *pull/need* defines all tasks and their timing.
5. **Pursue perfection.** Constantly improve, and make all imperfections visible to all, which is motivating to the continuous process of improvement.
6. **Respect for people.** Create a system of mutually respectful, trusting, honest, cooperating, and synergistic relationships of key stakeholders, motivating staff to exhibit top capabilities.

Lean Enablers for SE

LEfSE is a major product recently released in the field of Lean SE. It is a comprehensive checklist of 194 practices and recommendations formulated as the *dos and don'ts* of SE, containing tacit knowledge (collective wisdom) on how to prepare for, plan, execute, and practice SE and related enterprise management using Lean Thinking. Each enabler enhances the

program value and reduces some waste. As a set, the enablers are focused on providing more affordable solutions to increasingly complex challenges and improving response time from the identification of need to the release of the system. The enablers deal with mission assurance and promote practices that optimize workflow and reduce waste.

The enablers are formulated as a Web-based addendum to the traditional SE manuals—such as "The International Council on Systems Engineering [INCOSE] Handbook," ISO 15288, and similar NASA, DoD, or company manuals—and do not repeat the practices made therein, which are regarded as sound.

The LEfSE practices are organized into the previously mentioned six Lean Principles. The practices cover a large spectrum of SE and other relevant enterprise management practices, with a general focus to improve program value and stakeholder satisfaction, and reduce waste, delays, cost overruns, and frustrations[3]. The full text of the LEfSE is too long for the present article, therefore only a brief summary is given herein. The full text is available online[4].

- Under the **Value Principle**, the enablers promote a robust process of establishing the value of the end-product or system to the customer with crystal clarity. The process should be customer-focused, involving the customer frequently and aligning the enterprise employees accordingly.
- The enablers under the **Value Stream Principle** emphasize waste-preventing measures, solid preparation of the personnel and processes for subsequent efficient workflow and healthy relationships between stakeholders (customer, contractor, suppliers, and employees); detailed program planning; frontloading; and use of leading indicators and quality metrics.
- The **Flow Principle** lists the enablers which promote the uninterrupted flow of robust quality work and first-time right; steady competence instead of hero behavior in crises; excellent communication and coordination; concurrency; frequent clarification of the requirements; and making program progress visible to all.
- The enablers listed under the **Pull Principle** are a powerful guard against the waste of rework and overproduction. They promote pulling tasks and outputs based on need (and rejecting others as waste) and better coordination between the pairs of employees handling any transaction before their work begins (so that the result can be first-time right).
- The **Perfection Principle** promotes excellence in the SE and enterprise processes; the use of the wealth of lessons learned from previous programs in the current program; the development of perfect collaboration policy across people and processes; and driving out waste through standardization and continuous improvement. A category of these enablers calls for a more important role of systems engineers, with RAA for the overall technical success of the program.
- Finally, the **Respect-for-People Principle** contains enablers that promote the enterprise culture of trust, openness, respect, empowerment, cooperation, teamwork, synergy, good communication and coordination; and enable people for excellence.

"The practices cover a large spectrum of SE ... with a general focus to improve program value and stakeholder satisfaction, and reduce waste, delays, cost overruns, and frustrations."

LEfSE were developed by 14 experienced practitioners organized into two teams, some recognized leaders in Lean and System Engineering from industry, academia and governments (from the U.S., United Kingdom, and Israel), with cooperation from the 100-member strong international LSE WG of INCOSE [11].

Both SE and Lean represent challenging areas for research as they are grounded in industrial and government practice rather than laboratory work or theory. It is well-known that hard data about SE in large programs is difficult to obtain because:

- The programs are classified and proprietary.
- The companies are not willing to release such data even when it exists.
- In many cases, the data is non-existent, of a poor quality, lacks normalization, suffers from discontinuities over long program schedules, and is convoluted with other enterprise activities.

As a result, it is difficult to collect the data needed to perform hypothesis testing. Therefore, rather than to rely on explicit program data, the enablers were developed from collective tacit knowledge, wisdom, and experience of the LSE WG members. Such an approach has been practiced for ages by numerous institutions, and is being described in [12]. LEfSE have been formulated for industry SE practitioners, but the development benefited from academic depth, breadth, and rigor; the latter emphasis provided by surveys and benchmarking to published data, as follows.

The development of LEfSE included five phases: Conceptual, Alpha, Beta, Prototype, and Version 1.0. It was evaluated by separate surveys in the Beta and Prototype phases and by comparisons with the recent programmatic recommendations by GAO and NASA [1, 2, 3, 4]. The surveys indicated that LEfSE are regarded as important for program success but are not widely used by industry. The comparisons indicated that LEfSE are consistent with the NASA and GAO recommendations, but are significantly more detailed and comprehensive.

Intended Use

The LEfSE are not intended to become a mandatory tool. Instead, they should be used as a checklist of good holistic practices. Some are intended for top enterprise managers, some for programs, and others for line employees. Some are more actionable than others, and some are easier to implement than others. Some enablers may require changes in company policies and culture. However, employee awareness of even those least actionable and most difficult to implement enablers should improve the thinking at work.

The creators believe that as many systems (and other) engineers, enterprise managers, and customer representatives as possible should be trained in the LEfSE, as it will lead to better programs. At this time, a large effort of offering tutorials and lectures about the LEfSE throughout INCOSE chapters, industry, and academia is ongoing.

The published product includes examples of the programs and companies that practice the given enablers. Also listed is the average value measuring the use of a given enabler in industry, obtained from the surveys.

A formal online process of continuous improvement and periodic new releases of

the LEfSE has been set up as new knowledge and experience becomes available. A comprehensive description of the history of LSE, the development process of LEfSE, the full text of the enablers, the surveys, and industrial examples can be found in [11]. ◆

References

1. GAO. "Defense Acquisitions – Assessments of Selected Weapon Programs." GAO-07-4065SP. Washington, D.C. Mar. 2008 <www.gao.gov/new.items/d08467sp.pdf>.
2. GAO. "Best Practices – Increased Focus on Requirements and Oversight Needed to Improve DOD's Acquisition Environment and Weapon System Quality." GAO-08-294. Washington, D.C. Feb. 2008 <www.gao.gov/new.items/d08294.pdf>.
3. GAO. "Space Acquisitions – Major Space Programs Still at Risk for Cost and Schedule Increases." GAO-08-552T. Washington, D.C. 4 Mar. 2008 <www.gao.gov/new.items/d08552t.pdf>.
4. "NASA Pilot Benchmarking Initiative: Exploring Design Excellence Leading to Improved Safety and Reliability." Final Report, Oct. 2007.
5. LAI. "Phase I." 1 Jan. 2009 <http://lean.mit.edu/index.php?/about-lai/history/phase-one.html>.
6. McManus, Hugh L. "Product Development Value Stream Mapping Manual." LAI Release Beta, MIT, Apr. 2004.
7. Slack, Robert A. "Application of Lean Principles to the Military Aerospace Product Development Process." Master of Science – Engineering and Management Thesis, MIT, Dec. 1998.
8. Oppenheim, Bohdan W. "Lean Product Development Flow." Journal of Systems Engineering. Vol. 7, No. 4, 2004.
9. Womack, James P., and Daniel T. Jones. Lean Thinking. New York: Simon & Schuster, 1996.
10. Morgan, James M. and Jeffrey K. Liker. The Toyota Product Development System – Integrating People, Process And Technology. New York: Productivity Press, 2006.
11. Oppenheim, Bohdan W., Earl M. Murman, and Deborah Secor. "Lean Enablers for Systems Engineering." Submitted to Journal of Systems Engineering. Dec. 2008.
12. Webb, Luke. "Knowledge Management for Through Life Support." Doctoral Thesis (in progress) via private communication. RMIT University (Australia), 2008.

About the Author

Bohdan W. Oppenheim, Ph.D., is a professor of mechanical and systems engineering at Loyola Marymount University. He is the founder and co-chair of the LSE WG of INCOSE and serves as the local coordinator of the Lean Aerospace Initiative Educational Network. Oppenheim has worked for Northrop, the Aerospace Corporation, and Global Marine, and has served as a Lean consultant for Boeing and 50 other firms. He has a doctorate in dynamics from the University of Southampton (U.K.), a naval architect's degree from MIT, a master's degree in ocean systems from the Stevens Institute of Technology, and a bachelor's degree in mechanical engineering and aeronautics from Warsaw Technical University. Oppenheim is a member of INCOSE and is a fellow of the Institution for the Advancement of Engineering.

Loyola Marymount University
Pereira Hall of Engineering
RM 204
Los Angeles, CA 90045
Phone: (310) 338-2825
Fax: (310) 338-6028
E-mail: boppenheim@lmu.edu

Notes

1. The early use of the term LSE is sometimes met with concern that this might be a "re-packaged faster, better, cheaper" initiative, leading to cuts in SE at a time when the profession is struggling to increase the level and quality of SE effort in programs. Our work clearly disproves this concern.
2. The original formulation had five principles; the sixth (the Respect-for-People Principle) was added at a later time.
3. LEfSE practices do not deal, however, with explicit financial steps such as cost estimating or earned value analysis, which are regarded as a separate activity.
4. A PowerPoint presentation is available in PDF format at: <http://cse.lmu.edu/Assets/Colleges+Schools/CSE/Lean+Enablers+for+SE+Version+1.01.pdf>.

A Framework for Systems Engineering Development of Complex Systems

Dr. Karl L. Brunson, Dr. Jeffrey Beach, Dr. Thomas A. Mazzuchi, and Dr. Shahram Sarkani
George Washington University

In developing complex systems, evaluating potential schedule and cost risks is essential. With the Incremental Commitment Model (ICM), schedule tasks can be evaluated against manufacturing and technology risk drivers. In this article, these risk drivers are analyzed using a comprehensive approach with emphasis being placed on quantitative risk analysis through Monte Carlo simulation. Through modeling the behavior of a hypothetical project schedule for a notional spacecraft system, the authors show how the ICM framework is implemented in complex system development. The result is a repeatable, inherent, risk-driven commitment process that can stabilize and synchronize systems engineering and acquisition processes.

The level of complexity needed to develop spacecraft systems and other emerging technologies require programs to develop risk management and risk planning techniques that can potentially identify schedule and cost risks as early as possible during the acquisition life cycle. According to the Government Accountability Office (GAO), studies have shown that there has been an increase in schedule and cost overruns that involve complex systems and emerging technologies. This is often the case when projects exceed scheduled activity durations, resulting in frequent budget overruns. There are a plethora of risks that factor into inaccurate schedule estimates, including the elusive emerging requirements to the lack of process understanding.

Unfortunately, it is common to observe how requirements established during the earlier stages of an acquisition life cycle are changed to accommodate customer requests, thus impacting schedule and delivery costs. These impacts invariably affect scheduled activities from the design through the development and production of the complex system. To improve implementation and the understanding of the life-cycle processes for the complex system, it is essential that the development of a work breakdown structure (WBS), or an architecture and its interfaces within the appropriate hierarchical levels of decomposition, be accurately structured.

The structure of this architecture coincides with the development and implementation of activities that are required for the design, development, and production stages of the life cycle. As a result, the developed schedule activities influence the cost of delivery. Since scheduled activities impact the cost to develop complex systems, it can be shown that there is an inherent relationship between the complex systems architecture and the process activities required for schedule and cost estimating. As a result, project schedule and cost estimation play an important role in driving key acquisition life-cycle decisions.

Developing and delivering complex systems requires the management of complex risks such as uncertainty usage, schedule uncertainties, uncertainties associated with technology maturity, manufacturing maturity, technical design, and technical complexity [1]. Acquisition life-cycle decisions can be potentially flawed if the systems engineering development model isn't appropriately matched to the complex system being developed. To address the challenge of selecting a candidate systems engineering development model, the Committee on Human-System Design Support for Changing Technology recommended implementation of the ICM as a reasonably robust framework for the "progressive reduction of risk through the full life cycle of system development, to produce a cost-effective system that meets the needs of all the stakeholders" [1]. The ICM integrates key strengths or attributes of other models into an integrated framework while introducing risk decision anchor points throughout the life cycle [2]. To complement implementation of this model, our research provides a hypothetical example that incorporates maturity risk drivers—technology readiness levels (TRL) and manufacturing readiness levels (MRL)—within a notional ICM framework as an approach to assess schedule and cost risks during the development of a complex system.

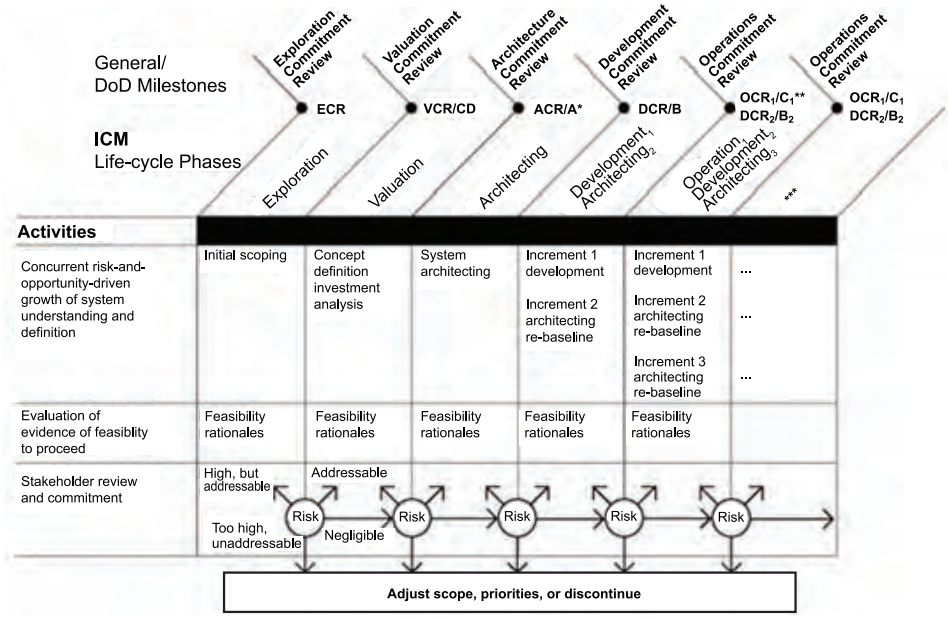
> *"There are a plethora of risks that factor into accurate schedule estimates ..."*

The method chosen to evaluate schedule and cost risk drivers for this research is known as Monte Carlo simulation. This method is used to model probabilistic behaviors of activities throughout the acquisition life cycle. The term *Monte Carlo* has been used interchangeably with probabilistic simulation since it is a technique used to randomly select numbers from a probability distribution or sampling. For our work, Monte Carlo simulation modeled the behaviors of schedule and cost uncertainties while providing traceability and consequence between the risk drivers and activities within the acquisition life cycle. Specifically, the simulation modeled the behaviors of tasks and activities derived from the spacecraft systems (the WBS) after mapping various risk drivers to those tasks [3]. This allows for exploration of the likelihood and consequences that these risk drivers have scheduled task activities throughout the acquisition life cycle.

The ICM Framework

The ICM was developed to ensure the flexibility of implementing one or more frameworks throughout each stage of an acquisition life cycle. The model was built upon five key principles that are critical for system development: 1) customer satisfaction; 2) incremental growth of system definition and stakeholder commitment; 3) iterative system definition and development; 4) concurrent system definition and development; and 5) management and project risk [2]. These principles are also proven strengths of other models such as the Waterfall, iterative, Rational Unified Process (RUP), and spiral development frameworks.

The ICM is unique in that it merges these key principles into one framework [2] to provide a process model that is robust with a central focus to progressively reduce potential risks throughout the entire life cycle, culminating in the development of



* A, B, and C represent milestones, as defined in DoD Instruction 5000.2.
** The subscripts represent their increment number, either Increment 1, 2, or 3.
*** Data repeated from previous column.
CD = Concept Definition.

Figure 1: *ICM and DoD Milestone Traceability*

complex systems that are both schedule- and cost-effective. The ICM framework also provides the decision-maker with the flexibility to recognize potential risks that coincide with system's maturity and complexity of scope. To accommodate the ensuing consequences of potential risk drivers, the ICM framework implements a risk management principle that associates risk-driven tasks and activities for each stage of the acquisition life cycle.

The construct of the ICM framework is comprised of two major stages. Stage I, the Incremental Definition, entails the initial design stages of the system where the conceptual definition and feasibility studies are conducted for a better understanding of the system and stakeholder commitment. Stage II, Incremental Development and Operations, is where the increments of operational capability are developed and integrated into schedules that correlate to the development and evolution of the complex system. The activities within each stage are risk-driven to account for process agility and rigor to ensure that the system objectives are met throughout the systems development life cycle [2]. The concepts that the ICM framework is built upon include:

- Early verification and validation concepts of the V-model.
- Concurrency concepts of the concurrent engineering model.
- Concepts from Agile and Lean models.
- Risk-driven concepts of the spiral model.
- Phases and anchor points of the RUP.
- Systems of systems acquisition concepts of the spiral model.

Synergistic structuring of one or more process models within the ICM framework provides the tailoring flexibility to accommodate the varying maturity characteristics of any complex system; Barry Boehm and Jo Ann Lane provide a more detailed discussion regarding the ICM framework in [2]. An illustration of an integrated DoD/ICM life-cycle framework is provided in Figure 1. This view aligns milestones A, B, and C, representing the designated commitment point of key stages. A more detailed discussion of anchor points can be found in both [1] and [4].

Our work focused on how the ICM framework is implemented in the development of a complex system by modeling the behavior of a hypothetical project schedule of a notional spacecraft system [3]. The modeling technique, Monte Carlo simulation, will be used to help the decision-maker evaluate schedule durations and cost estimates that are impacted by risk drivers. It will also aid the decision-maker with establishing preliminary risk management assessments.

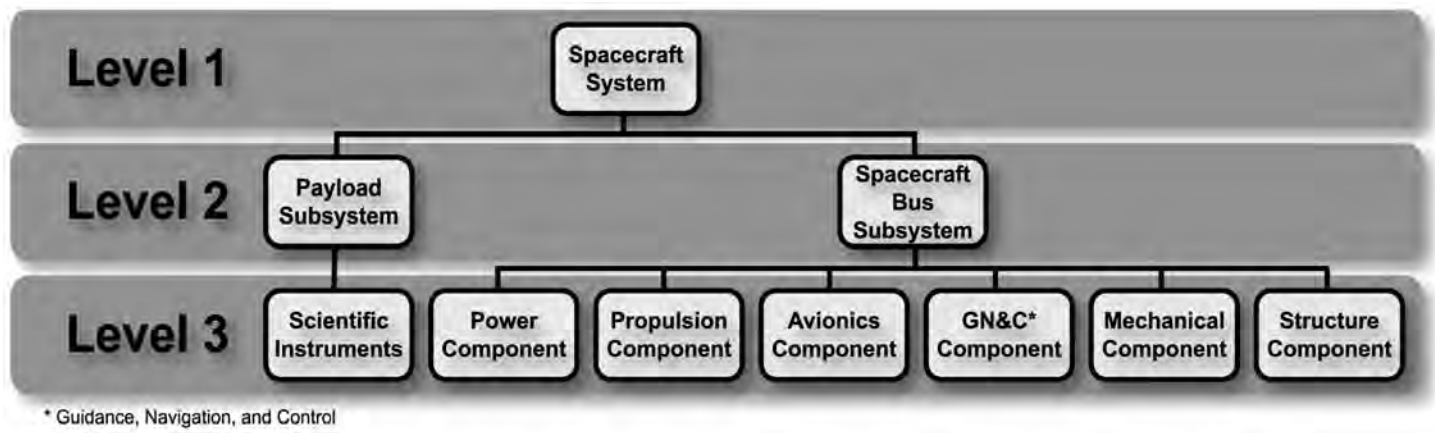
Spacecraft System WBS

The WBS shown in Figure 2 illustrates how the notional spacecraft system is defined, developed, and maintained throughout the acquisition life cycle [4]. The hierarchical breakdown of the spacecraft is used to understand the products contained within each level of decomposition. The hierarchical levels that comprise the products provide the work structure necessary to develop tasks and activities throughout the acquisition life cycle. The products include hardware, software, documents, and processes. However, the WBS in this case is only focused on the hardware of the spacecraft at a third level of decomposition. Each level of decomposition was used to identify all of the subsystems and components that influence the preceding level and are aggregated to the top of the hierarchy through functional relationships.

Spacecraft System Project Schedule

The spacecraft system's project schedule

Figure 2: *WBS of Notional Spacecraft System*



ID	Description	Rem Duration	Start	Finish	Minimum Duration	Most Likely	Maximum Duration	Total Cost	Duration Uncertainty
S010	Project Start	0	12-Oct-05	11-Oct-05					
S020	Exploration and Valuation	60	12-Oct-05	3-Jan-06				$565,000	
S030	Valuation Commitment and Agreement	10	12-Oct-05	25-Oct-05	9	10	11	$75,000	L
S040	Concept Studies	15	26-Oct-05	15-Nov-05	14	15	20	$240,000	M
S050	Concept and Technology Develpoment	50	26-Oct-05	3-Jan-06	48	60	75	$250,000	H
S060	Architecting	225	4-Jan-06	14-Nov-06				$9,691,865	
S070	Initial Spacecraft Bus Design	40	4-Jan-06	28-Feb-06	36	40	44	$2,456,490	L
S080	Initial Spacecraft Payload Design	22	4-Jan-06	2-Feb-06	19	22	36	$275,000	L
S090	Architecture Commitment	10	1-Mar-06	14-Mar-06	9	10	13	$50,000	M
S100	Design Power Component	81	1-Mar-06	21-Jun-06	61	81	101	$526,500	M
S110	Design Propulsion Component	120	15-Mar-06	29-Aug-06	108	120	132	$1,020,000	L
S120	Design Avionics Component	76	1-Mar-06	14-Jun-06	57	76	95	$798,000	M
S130	Design GN&C	60	1-Mar-06	23-May-06	54	60	66	$750,000	L
S140	Design Mechnical Component	80	15-Mar-06	4-Jul-06	76	96	120	$1,145,875	H
S150	Design Structure Component	54	15-Mar-06	29-May-06	49	54	70	$945,000	M
S160	Design Payload Subsystem	83	3-Feb-06	30-May-06	62	83	104	$1,037,500	M
S170	Final Design	55	30-Aug-06	14-Nov-06	52	66	83	$687,500	H
S180	Development and Architecting (1)	465	12-Oct-05	24-Jul-07				$9,867,760	
S190	Development Commitment	10	12-Oct-05	25-Oct-05	8	10	13	$75,000	L
S200	Fabricate Spacecraft Payload	80	15-Nov-06	6-Mar-07	72	80	104	$5,134,560	M
S210	Fabricate Spacecraft Bus	110	15-Nov-06	17-Apr-07	99	110	143	$3,338,000	M
S220	Assemble	30	13-Jun-07	24-Jul-07	29	36	45	$1,320,200	H
S230	Development and Architecting (2)	150	7-Mar-07	2-Oct-07				$5,198,030	
S240	Test Spacecraft Payload	35	7-Mar-07	24-Apr-07	32	35	46	$1,809,500	M
S250	Test Spacecraft Bus	40	18-Apr-07	12-Jun-07	38	48	60	$732,400	H
S260	Final Testing	70	25-Jul-07	2-Oct-07	63	70	77	$2,656,130	L
S270	Delivery	10	3-Oct-07	16-Oct-07					
S280	Delivery	10	3-Oct-07	16-Oct-07	9	10	11		L
S290	Project Completion	0	17-Oct-07	16-Oct-07					

Table 1: *Sample Spacecraft Project Schedule With Notional ICM Framework*

was used to provide key activity dates and durations that are associated with the products of the WBS. The WBS sets the foundation of all scheduled activities, thus impacting cost estimates. The duration of the scheduled activities was driven by many factors such as the technical complexity of work to be performed, manufacturing availability of components to be developed, and the technical maturity of components to be assembled. These factors were risk drivers that had an effect on the uncertainties of the project schedule.

The project schedule illustrates two key elements: the influence that risk drivers have on schedule tasks, and the influence that tasks and activities have on each other through precedence relationships. For example, let's say that Task 1 must end before Task 2 can begin or Task 3 cannot begin until Tasks 1 and 2 have ended, respectively [5]. Because of the shared interrelationships of the schedule's tasks, it is inevitable that any overrun in scheduled activities will most likely impact the duration of other tasks and activities of the project schedule—thus increasing the likelihood of cost overrun. Table 1 illustrates a sample project schedule for the development of the spacecraft system implementing the ICM framework.

Risk Management

To evaluate the proposed ICM framework effectively, it is important to organize a risk management approach that ensures the identification and quantification of risks and uncertainties that may impact a project's schedule and cost [6]. Because of the increasing complexity of the development of spacecraft systems, it is likely that a project's schedule tasks are ultimately interrelated and associated with cost [7]. In the context of system development programs, schedule and cost risks may determine whether or not the program will complete the systems development on schedule and on budget. If the program successfully meets or exceeds the schedule and budget expectations of the customer, it will likely be due to the effective implementation of risk management processes [8].

The ICM model implements risk management anchor points throughout each stage of the life cycle in order to improve the possibility of success for the development of the complex system [2]. The project schedule of the notional spacecraft system illustrated similar risk anchor points and was modeled with the appropriate stage tasks and activities against associated risks. Cost and schedule risks co-exist because of inherent uncertainties regarding the time and costs required to complete tasks of a project's schedule [9]. To understand the uncertainties of schedule and cost risks, a risk mitigation strategy must be implemented to minimize the impact of these risks.

The risk mitigation strategy to be implemented with our research includes the following steps:

1. **Risk Identification.** Evaluation of a risk's probability of occurrence and the impacts or consequence of risks against schedule and costs.
2. **Risk Assessment.** Quantification of the information acquired from risk identification to assess project schedule, cost, and technical risks.
3. **Risk Analysis.** Quantification of risk data in terms of probability of occurrence and the eventual consequence(s) if a risk does occur.
4. **Risk Mitigation.** Determination of actions to be implemented to reduce schedule and cost risks.

These steps are supported by empirical data that show how cost estimates are often linked to activity durations via schedule risk results. This is often seen when schedule risk analysis results are used as input to cost risk analysis and is primarily implemented to identify the uncertainty in activity durations in order to assess cost risks [3].

Risk Identification

The risk identification process begins by evaluating key risks (independent variables) and their respective uncertain impacts, throughout the acquisition life cycle, on the project's schedule and cost variables. The risk identification process is implemented by a team of experts who evaluate project tasks and activities against the categorized risks that have varying probabilities of occurrences within each stage of the life cycle. It should also be noted that the risk identification process can be implemented with empirical data for the complex system being developed.

Our research identified maturity risks (TRL and MRL) as technical risks to be evaluated against project events. A brief description for TRL and MRL ratings are provided in Table 2 (see next page) [10]. The risks were evaluated independently against tasks and activities that would

impact subsystems and components of the spacecraft's WBS. This approach allows a quick and simple explanation of how these risk drivers impact the final schedule and cost results of the project schedule.

The maturity risk drivers (TRL and MRL) are independent variables that carry a correlating metric weighting system that establishes the readiness to implement based upon factors such as the stability, technical complexity, and maturity of a spacecraft's systems and subsystems. Had the research been more detailed, the results of the weighting metric system would be used to establish an entrance and exit criteria for identifying the most appropriate framework or framework combinations to be implemented throughout the life cycle. However, that step was bypassed for this study; the given ICM framework was considered baseline for a nominal matured spacecraft system. The hypothetical impact for both TRL and MRL risk drivers are shown in Figure 3 with a brief explanation. The example provided for that figure summarizes the ratings for each risk driver in a pre- and post-mitigated format. However, the focus of this research is to demonstrate how a system or subsystem may have different maturity ratings throughout the WBS and illustrate the impacts on tasks and activity durations.

Figure 3 illustrates the process used to identify risks that affect the project schedule (generated using [5]). Each risk is evaluated against a task where they're rated based upon the probability of occurrence and their *severity effects* on schedule and cost. This example demonstrates a pre-and-post mitigation evaluation—quantified after running the Monte Carlo simulation—to address the schedule's behavioral uncertainties. The first step in the Monte Carlo simulation development required a qualitative risk assessment of the defined risk drivers. The qualitative risk assessment is performed with the use of the WBS and the identification of an expert familiar with the tasks and activities of the development and production phases. The input from the expert will be used to establish the qualitative risk assessment with the following important steps:

1. Develop a working list of risk drivers that pose a threat to the project schedule, cost, or development performance.
2. Develop a risk ranking guide that establishes the probability of risk occurrence. This study uses very low (VL), low (L), medium (M), high (H), very high (VH) probabilities of occurrence for activities and tasks of the WBS affecting schedule and cost.
3. Identify the impacts or consequences of the risk drivers by evaluating the probability of occurrence and the magnitude of impact on the schedule, cost, and development performance. Establish qualitative descriptions to identify the assessed risk drivers with the use of a risk matrix.
4. Establish a final risk score for each risk driver after completing steps 1-3, followed by populating the risk matrix to illustrate the magnitude of all identified risks. Figure 3 illustrates how steps 1-4 were implemented and ranked for MRL-1.

Table 2: *TRL and MRL Rating Descriptions and Relationships*

TRL	Description	MRL	Description
1	Basic principles observed and reported.	1	Manufacturing concepts identified.
2	Technology concept and/or application formulated.	2	Manufacturing concepts identified.
3	Analytical and experimental critical function and/or characteristic proof of concept.	3	Manufacturing concepts identified.
4	Component and/or breadboard validation in a laboratory environment.	4	System, component, or item validation in a laboratory environment.
5	Component and/or breadboard validation in a relevant environment.	5	System, component, or item validation in initial relevant environment. Engineering application, breadboard, brassboard development.
6	System/subsystem model or prototype demonstration in a relevant environment.	6	System, component, or item in prototype demonstration beyond breadboard, brassboard development.
7	System prototype demonstration in an operational environment.	7	System, component, or item in advanced development.
8	Actual system completed and qualified through test and demonstration.	8	System, component, or item in advanced development. Ready for Low Rate Initial Production (LRIP).
		9	System, component, or item previously produced or in production or the system, component, or item is in LRIP. Ready for Full Rate Production (FRP).
9	Actual system proven through successful mission operations.	10	System, component, or item previously produced or in production or the system component or item is in low rate initial productions.

Risk Assessment

Risk assessment is the process of classifying risks into categories characterized by their frequency of occurrence and the severity of their consequences. The risk assessment can be performed through either qualitative or quantitative evaluations as well as through a comprehensive evaluation combining both assessment types. Qualitative risk assessment is considered to be the process of prioritizing risks based upon the risk's probability of occurrence ranging from unlikely to most likely. The second aspect of the qualitative risk assessment is when the risks are prioritized based upon the risk's *severity of consequence*. Quantitative risk assessment is considered to be the process of prioritizing risks using statistical techniques to estimate the project's numerical outcome (schedule/cost behaviors) based upon identified project risks through the use of probability distributions. Monte Carlo simulation is commonly used to model project behavior [11].

For our research, the quantitative risk

Risk			Pre-mitigation					Response			Post-mitigation				
ID	Type	Title	Probability	Schedule	Cost	Performance	Score	Type	Title	Total Cost	Probability	Schedule	Cost	Performance	Score
1 - MRL	Threat	Poor understanding of Manufacturing Concepts	VL	M	L	VL	2	Reduce	Introduce penalties for design changes	$10,000.00	L	L	L	VL	3
2 - TRL	Threat	Technology Concept Not Fully Defined	L	M	M	VH	24	Reduce	Improve Technology Concept	$55,000.00	N	VH	VH	VH	0
3 - TRL	Threat	Proof of Concept Delay	M	M	H	H	20	Reduce	Clear definition of Proof of Concept	$500,000.00	L	M	L	H	12
4- MRL	Threat	Key resource unavailable	VL	M	H	H	4	Reduce	Change resource assignment policy	$300,000.00	VL	L	L	L	1
5 - TRL	Threat	Sub-system not validated	VL	VL	L	N	1	Reduce	Implement CMMI Early	$50,000.00	L	L	N	N	3
6 - TRL	Threat	Fabrication contractor cannot meet deliverable requirements.	VL	M	M	M	2	Reduce		$.00	N	M	M	M	0
7 - TRL	Opportunity	Sub-system prepared for early fabrication	VH	VL	N	N	5	Enhance	Exceed manufacting standards	$.00	N	M	M	L	0
8 - TRL	Threat	Testing fails	VL	VL	M	N	2	Reduce		$.00	VL	VL	L	N	1
9 - TRL	Opportunity	System completed and qualified	VH	VL	N	N	5	Enhance		$.00	N	N	N	N	0

Scenario Name	ID	Title	Probability	Mitigation Cost	Mitigation Actions
Pre-mitigation	1 - MRL	Poor understanding of Manufacturing Concepts	5.00%	$10,000.00	Introduce penalties for design changes
	2 - TRL	Technology Concept Not Fully Defined	20.00%	$55,000.00	Improve Technology Concept
	3 - TRL	Proof of Concept Delay	40.00%	$500,000.00	Clear definition of Proof of Concept
	4- MRL	Key resource unavailable	5.00%	$300,000.00	Change resource assignment policy
	5 - TRL	Sub-system not validated	5.00%	$50,000.00	Implement CMMI Early
	6 - TRL	Fabrication contractor cannot meet deliverable requirements.	5.00%	$.00	
	7 - TRL	Sub-system prepared for early fabrication	85.00%	$.00	Exceed manufacting standards
	8 - TRL	Testing fails	5.00%	$.00	
	9 - TRL	System completed and qualified	85.00%	$.00	
Post-mitigation	1 - MRL	Poor understanding of Manufacturing Concepts	20.00%	$10,000.00	Introduce penalties for design changes
	2 - TRL	Technology Concept Not Fully Defined	0.00%	$55,000.00	Improve Technology Concept
	3 - TRL	Proof of Concept Delay	20.00%	$500,000.00	Clear definition of Proof of Concept
	4- MRL	Key resource unavailable	5.00%	$300,000.00	Change resource assignment policy
	5 - TRL	Sub-system not validated	20.00%	$50,000.00	Implement CMMI Early
	6 - TRL	Fabrication contractor cannot meet deliverable requirements.	0.00%	$.00	
	7 - TRL	Sub-system prepared for early fabrication	0.00%	$.00	Exceed manufacting standards
	8 - TRL	Testing fails	5.00%	$.00	
	9 - TRL	System completed and qualified	0.00%	$.00	

Probability Scale					
	Very Low	Low	Medium	High	Very High
	Up to 10%	10% to 30%	30% to 50%	50% to 70%	70% or higher

Impact Scales and Types					
	Very Low	Low	Medium	High	Very High
Schedule*	Up to 10	10 to 20	20 to 50	50 to 150	150 or higher
Cost*	Up to $10,000	$10,000 to $50,000	$50,000 to $100,000	$100,000 to $500,000	$500,000 or higher

Probability and Impact Scoring					
	Impact Very Low	Low	Medium	High	Very High
Very High	5	9	18		
High	4	7	14		
Medium	3	5	10	20	
Low	2	3	6	12	
Very Low	1	1	2	4	8

Figure 3: *Risk Identification of MRL-1*

assessment was based upon the project schedule that was developed for the spacecraft system. The goal was to understand the levels of uncertainty inherent within tasks and activities of a project's schedule. These uncertainties (task durations) were evaluated by probability distributions via three-point estimates and are acquired either through empirical data or expert judgment. Duration values associated with each task or activity risk are represented by subjective bounds *least likely*, *most likely*, and *optimistic*, and are analyzed after the Monte Carlo simulation is run.

Monte Carlo simulation is designed to iterate the project schedule's tasks and activities multiple times by randomly selecting task or activity duration values for each iteration from the probability distribution type chosen. The outcome results of the simulation were then used to provide the possible end dates of all tasks and activities based upon the respective associated risk drivers for the spacecraft's project schedule.

Risk Analysis and Results

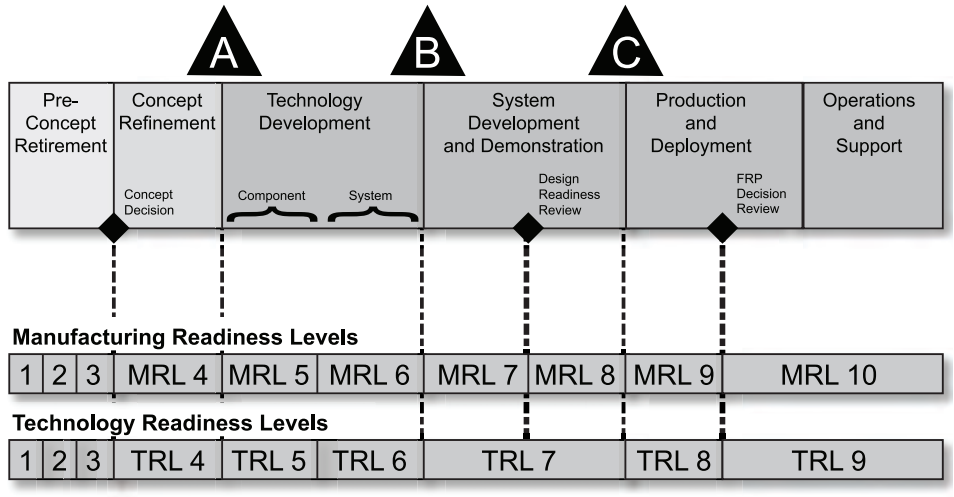
Risk analysis was conducted using Monte Carlo simulation to illustrate how the ICM framework can be implemented for the development of a spacecraft system [3]. The benefit of using this technique is that it generates schedule and cost estimates for uncertain input values through the use of probability distributions. It does this by randomly generating values and iteratively modeling the behavior of tasks and activities of a project schedule.

The methodology was used to demonstrate how TRL and MRL risk drivers can be mitigated within the ICM framework, and to understand and assess whether the project schedule will meet the required completion date without budget overruns. In order to properly model the behavior of the project schedule, the ICM framework was baselined with the DoD life-cycle stages [3]. Within each stage, planned tasks and activities were developed for a spacecraft's development and traced against the TRL and MRL risk drivers through risk identification. An example of the mapping process, illustrating the relationship of the risk drivers and the life-cycle stages, is provided in Figure 4.

Once the risk drivers were mapped to the appropriate activities, the uncertainty of the risks were pre- and post-mitigated through qualitative analysis with the options of using expert opinion and empirical data. Although our work was hypothetical, the data collection method included both expert opinion and empirical data. The probability distribution chosen to be simulated throughout the life cycle is frequently used to model expert opinion or empirical data. The distribution used was adapted so that the expert can provide three-point estimates that represent pessimistic, most likely, and optimistic inputs. The estimates correspond directly to the time estimates used as input variables for the Monte Carlo simulation. Therefore, the triangular distribution was chosen because it is the most commonly used distribution for modeling expert opinion and empirical data. The triangular distribution is also used when there is very little information known about the parameters outside the approximate estimate of its pessimistic, most likely, and optimistic variables. In addition, the uniform distribution was not chosen because it is known to be a very poor modeler of expert opinion and empirical data (since all of the values within its range have equal probability density). Thus, the density falls sharply to zero at the pessimistic and optimistic endpoint estimates.

The example (provided in Figure 3) identifies MRL-1 as the risk driver imposed upon the concept studies task of the Exploration and Valuation stages of the life cycle. The MRL-1 risk driver was then categorized and identified with a medium impact and low probability of occurrence. An illustration of the risk rating is provided in the risk matrix of Figure

Figure 4: *TRL and MRL Maturity Trace to DoD Acquisition Life Cycle*



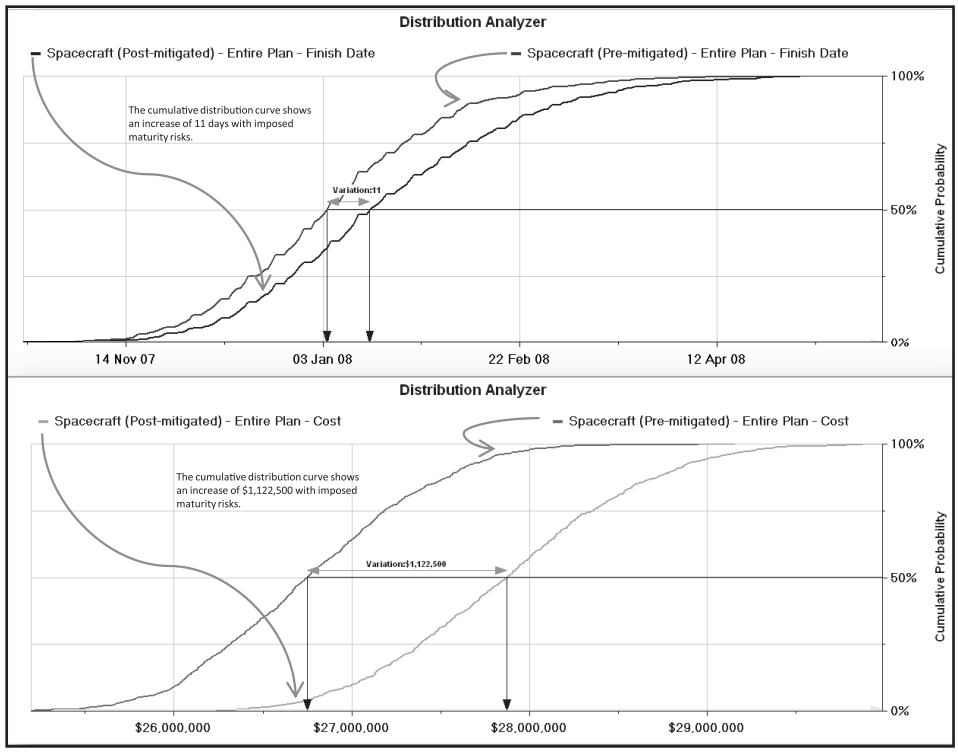


Figure 5: *Spacecraft Schedule and Cost Results of Non-Risk Imposed vs. Risk Imposed Simulations*

3, in addition to detailing pre- and post-mitigation definitions for that risk driver. The risk identification and mitigation process is a critical step in the event that contingency scenarios need to be formulated to successfully complete the project.

Now that all steps have been taken to develop the spacecraft's project model, the simulation is run and the results are evaluated to be realistic or unrealistic. After running the pre-mitigated project schedule, the statistical completion dates are provided in the project schedule results without the imposed maturity risks shown in Figure 5. The results of the spacecraft's project schedule were then used to establish the baseline model for the ICM acquisition life cycle. The next step, as outlined in the Risk Identification section, requires factoring the risk drivers into the project schedule's tasks and activities, followed by running another simulation to determine the final schedule outcome. The modeled behavior of the risk-driven schedule is provided in the project schedule with the imposed maturity risks of Figure 5.

Figure 5 also illustrates the spacecraft's cost result for non-risk- and risk-imposed simulations. The non-risk-imposed probability distributions for both schedule and cost outcome aided in the evaluation of the uncertainties of the maturity risk drivers that are incorporated into the risk-imposed results. Thus, it is clear that the risk-imposed simulation results have increased in cost when compared to the pre-mitigated results.

Risk Mitigation and Conclusions

The objective of this study was to evaluate the implementation of the ICM framework within a DoD acquisition life cycle while imposing TRL and MRL risk drivers. This was accomplished by the implementation of qualitative and quantitative statistical techniques and the use of Monte Carlo simulation to predict the probability of meeting the program's projected schedule and cost estimates. The project schedule activities developed for this study correlated to activities that comprised a notional ICM framework. As a result, the notional ICM framework established the baseline project schedule to be evaluated against technology and manufacturing maturity throughout the system development life cycle.

The evaluation was successfully performed using a step-by-step risk management process that was quantified through simulation. The triangular probability distribution was chosen to be used throughout the Monte Carlo simulation. This distribution type was chosen because three-point estimates (pessimistic, most likely, and optimistic) were used to represent workflow or activity durations of the life cycle. It should be noted that although life-cycle-critical paths were not identified and discussed in this study, the Monte Carlo simulation generated random variables to predict activity durations from each critical path probability distribution; this was ultimately used to develop the overall probability distribution of the system's project schedule. Because of the project schedule's simulation, insight was given to the decision-maker that revealed the consequences of the imposed maturity risks against the predicted schedule and cost estimates.

The project results contain qualitative (expert opinion) and quantitative (empirical) data. It is assumed that the degree of subjectivity associated with an expert's input is consistent with the expected data of the project schedule. However, if the subjective inputs are inaccurate, the results of the simulation can be very sensitive and reflect inaccurate schedule and cost estimates. Therefore, since the risk scores were developed by experts with some degree of subjectivity, it is important to consider evaluating the credibility of those experts in order to quantify their input. This evaluation can be performed with a technique called the classical method [12]. Although not used within our study, it is a credible approach to consider when validating subjective inputs.

The results of the Monte Carlo simulation demonstrated that the technology and manufacturing maturity risks influence the overall schedule and cost to develop the complex system; this is explicitly shown in the final schedule and cost results of Figure 5. The pre-mitigated status of the project schedule represents a simulation run where the maturity risk drivers are not applied to the activities of the project schedule. The post-mitigated status of the project schedule represents a simulation run where the maturity risk drivers are applied to the activities of the project schedule and a risk-mitigation strategy is developed but not implemented. This is consistent with the schedule slip and the cost increase illustrated in Figure 5. However, a simulation run that implements the risk-mitigated strategy should show improvement in the final project schedule and cost metrics. It should also be noted that the simulated results were used to illustrate two important elements:

1. There may be schedule and cost consequences when applying maturity risk drivers to the project schedule of the acquisition life cycle.
2. The analysis can provide a level of confidence in meeting the projected schedule and cost estimates of the overall project.

As a result of the risk management process developed for this study, decision-makers have a significant alternative mitigation strategy that can be implemented in order to minimize potential schedule and cost overruns.

For future research, this study estab-

lishes a framework for evaluating the impacts of maturity risks against schedule and cost and produces results that quantify a hypothetical baseline ICM framework. It also establishes the risk assessment approach to quantitatively evaluate and compare the metrics of other life-cycle models, further identifying the strengths and tailorability of the ICM framework.◆

References

1. Pew, Richard W., and Anne S. Mavor, eds. Human-System Integration in the System Development Process. Washington, D.C.: The National Academies Press, 2007.
2. Boehm, Barry, and Jo Ann Lane. "Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering." CROSSTALK Oct. 2007.
3. Brunson, Karl L., et al. "Toward a Framework for Implementing Systems Engineering Development for Complex Systems." Proc. of the 23rd International Forum on COCOMO and Systems/Software Cost Modeling and ICM Workshop 3. Los Angeles: University of Southern California, 2008.
4. DoD. MIL-HDBK-881A. Washington, D.C.: DoD, 2005.
5. Primavera Pertmaster Software. "Pertmaster Tutorial." Primavera Pertmaster, 2008.
6. van Dorp, Johan R., et al. "A Risk Management Procedure for the Washington State Ferries." Risk Analysis 21.1 (2001): 127-142.
7. Loureiro, Geilson, Paul G. Leaney, and Mike Hodgson. "A Systems Engineering Framework for Integrated Automotive Development." Systems Engineering 7.2 (2003): 153-167.
8. Robinette, G. Jeffrey, and Janet S. Marshall. "An Integrated Approach to Risk Management and Risk Assessment." Incose Insight 4.1 (2001).
9. GAO. Space Acquisitions DoD Needs to Take More Action to Address Unrealistic Initial Cost Estimates of Space Systems. Washington D.C.: GAO, 2006.
10. DoD. Technology Readiness Assessment Deskbook. Washington, D.C.: Deputy Under Secretary of Defense for Science and Technology, 2005.
11. NASA. Exploration Systems Risk Management Plan. Risk Management Plan. Washington, D.C.: NASA, 2007.
12. Bedford, Tim, and Roger M. Cooke. Probabilistic Risk Analysis: Foundations and Methods. New York: Cambridge University Press, 2006.

About the Authors

Karl L. Brunson, Ph.D., is currently senior staff aerospace engineer at Lockheed Martin where his professional experience spans modeling and simulation through systems engineering and architecture development. He received a master's degree in aerospace engineering with a specialization in space systems from the University of Maryland, and a bachelor's degree in aerospace engineering from Mississippi State University. Brunson recently completed his doctorate at George Washington University in systems engineering.

Lockheed Martin Corporation
13200 Woodland Park RD
STE 9066
Herndon, VA 20171
Phone: (703) 984-7229
Fax: (703) 984-7393
E-mail: karl.l.brunson@lmco.com

Thomas A. Mazzuchi, D.Sc., earned his doctoral and master's degrees in operations research at George Washington University, and a bachelor's degree in mathematics at Gettysburg College. He is chair of the Department of Engineering Management and Systems Engineering at George Washington University where he is also professor of both operations research and engineering management.

School of Engineering
and Applied Science
The George Washington University
Dept. of Engineering Management
and Systems Engineering
1776 G ST NW
STE 101
Washington, D.C. 20052
Phone: (202) 994-7424
Fax: (202) 994-0245
E-mail: mazzu@gwu.edu

Jeffrey Beach, D.Sc., is the head of the Structures and Composites Department at the Carderock Division of the Naval Surface Warfare Center, where he has worked since 1969. He received his bachelor's and master's degrees in aerospace engineering from the University of Maryland, and received his doctorate in engineering management and systems engineering from George Washington University.

School of Engineering
and Applied Science
The George Washington University
Dept. of Engineering Management
and Systems Engineering
20101 Academic WY
STE 227-A
Ashburn, VA 20147
Phone: (703) 726-8260
Fax: (703) 726-8251
E-mail: beachje@gwu.edu

Shahram Sarkani, Ph.D., is faculty adviser and head of George Washington University's (GWU) engineering management and systems engineering off-campus programs office. He has served as professor of engineering management and systems engineering since 1999, and is the founder and director (since 1993) of GWU's Laboratory for Infrastructure Safety and Reliability. Sarkani received his doctorate in civil engineering from Rice University, and his master's and bachelor's degrees in civil engineering from Louisiana State University.

School of Engineering
and Applied Science
The George Washington University
Dept. of Engineering Management
and Systems Engineering
2600 Michelson DR
STE 750
Irvine, CA 92612
Phone: (949) 724-9695
Fax: (949) 724-9694
E-mail: sarkani@gwu.edu

Why Software Requirements Traceability Remains a Challenge

Andrew Kannenberg
Garmin International

Dr. Hossein Saiedian
The University of Kansas

Why do so many challenges exist in traceability practices today? While many of these challenges can be overcome through organizational policy and procedure changes, quality requirements traceability tool support remains an open problem. After discussing the basics of software requirements traceability, this article shows why neither manual traceability methods nor existing COTS traceability tools are adequate for the current needs of the software engineering industry.

Software products are increasingly being deployed in complex, potentially dangerous products such as weapons systems, aircraft, and medical devices. Software products are critical because failure in these areas could result in loss of life, significant environmental damage, and major financial loss. This might lead one to believe that care would be taken to implement these software products using proven, reproducible methods. Unfortunately, this is not always the case.

In 1994, a Standish Group study [1] found that 53 percent of software projects failed outright and another 31 percent were challenged by extreme budget overruns. Since that time, many responses to the high rate of software project failures have been proposed. Examples include the SEI's CMMI®, the ISO's 9001:2000 for software development, and the IEEE's J-STD-016.

One feature that these software development standards have in common is that they all impose requirements traceability practices on the software development process. Requirements traceability can be defined as "the ability to describe and follow the life of a requirement, in both a forward and backward direction" [2]. This concept is shown in Figure 1.

Although many facets of a software project can be traced, the focus of this article is on requirements traceability; therefore, the term *traceability* is used to refer to requirements traceability throughout. See Figure 2, which provides an alternative view to Figure 1.

Research has shown that inadequate traceability is an important contributing factor to software project failures and budget overruns [3]. As a response, there has been an outpouring of research and literature on the subject of traceability, and many organizations have been striving to improve their traceability practices. These efforts have not been in vain. In 2006, The Standish Group updated their 1994 study [4], showing that only 19 percent of software projects failed outright with another 46 percent challenged by budget overruns. The improvement since 1994 is clearly shown in Table 1 (see page 16); however, room for growth remains.

Although the importance of traceability seems to be well-accepted in the software engineering industry, research suggests that many organizations still do not understand the principles of traceability and are struggling with implementing traceability practices in the software development life cycle [5]. Perhaps the biggest need is for a better understanding of why traceability is important and the challenges facing its implementation. This article attempts to address this need by studying the factors that make traceability important and discusses the challenges facing traceability practices in industry.

> *"Through traceability, each project engineer has access to contextual information that can assist them in determining where a requirement came from, its importance, how it was implemented, and how it was tested."*

The Importance of Traceability

Requirements traceability has been demonstrated to provide many benefits to organizations that make proper use of traceability techniques. This is why traceability is an important component of many standards for software development, such as the CMMI and ISO 9001:2000. Important benefits from traceability can be realized in the following areas: project management, process visibility, verification and validation (V&V), and maintenance [6].

Project Management

Traceability makes project management easier by simplifying project estimates. By following traceability links, a project manager can quickly see how many artifacts will be affected by a proposed change and can make an informed decision about the costs and risks associated with that change. Project managers can also utilize traceability to assist in measuring project progress. As requirements are traced to code and later to test cases, management can estimate the project completion status based on how many requirements have been traced to artifacts created later in the development cycle. This information can be used to estimate the schedule for a project during development and can be used to assess risk.

Process Visibility

Traceability offers improved process visibility to both project engineers and customers. Through traceability, each project engineer has access to contextual information that can assist them in determining where a requirement came from, its importance, how it was implemented, and how it was tested. Traceability can also be viewed as a customer satisfaction issue. If a project is audited or in the case of a lawsuit, traceability can be used to prove that particular requirements were implemented and tested. The availability of this information also increases customer confidence and satisfaction because it reassures customers that they will receive the product that they requested.

Verification and Validation

The most significant benefits provided by traceability can be realized during the V&V stages of a software project. Traceability offers the ability to assess system functionality on a per-requirement basis, from the

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

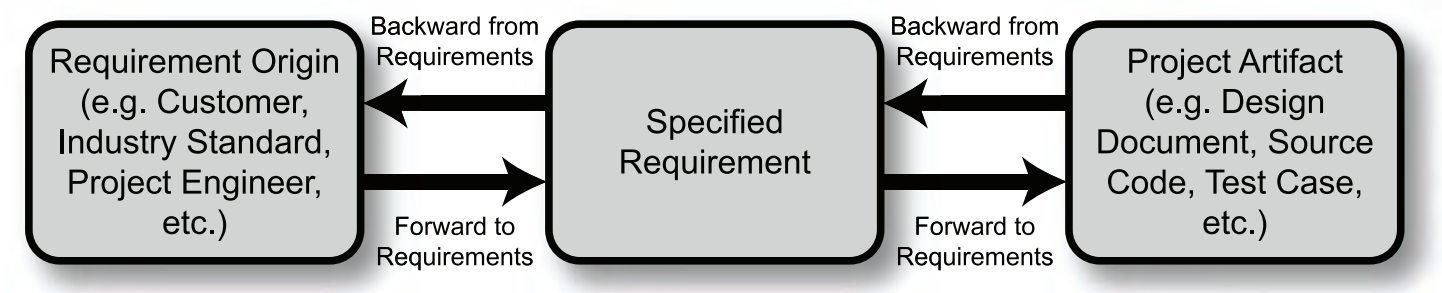


Figure 1: *A View of Software Requirements Traceability*

origin through the testing of each requirement. Properly implemented, traceability can be used to prove that a system complies with its requirements and that they have been implemented correctly. If a requirement can be traced forward to a design artifact, it validates that the requirement has been designed into the system. Likewise, if a requirement can be traced forward to the code, it validates that the requirement was implemented. Similarly, if a requirement can be traced to a test case, it demonstrates that the requirement has been verified through testing. Without traceability, it is impossible to demonstrate that a system has been fully verified and validated.

Maintenance

Traceability is also a valuable tool during the maintenance phase of a software project for many of the same reasons that it is valuable for project management. Initially defined requirements for a software project often change even after the project is completed, and it is important to be able to assess the potential impact of these changes. Traceability makes it easy to determine what requirements, design, code, and test cases need to be updated to fulfill a change request made during the software project's maintenance phase. This allows for estimates of the time and cost required to make a change.

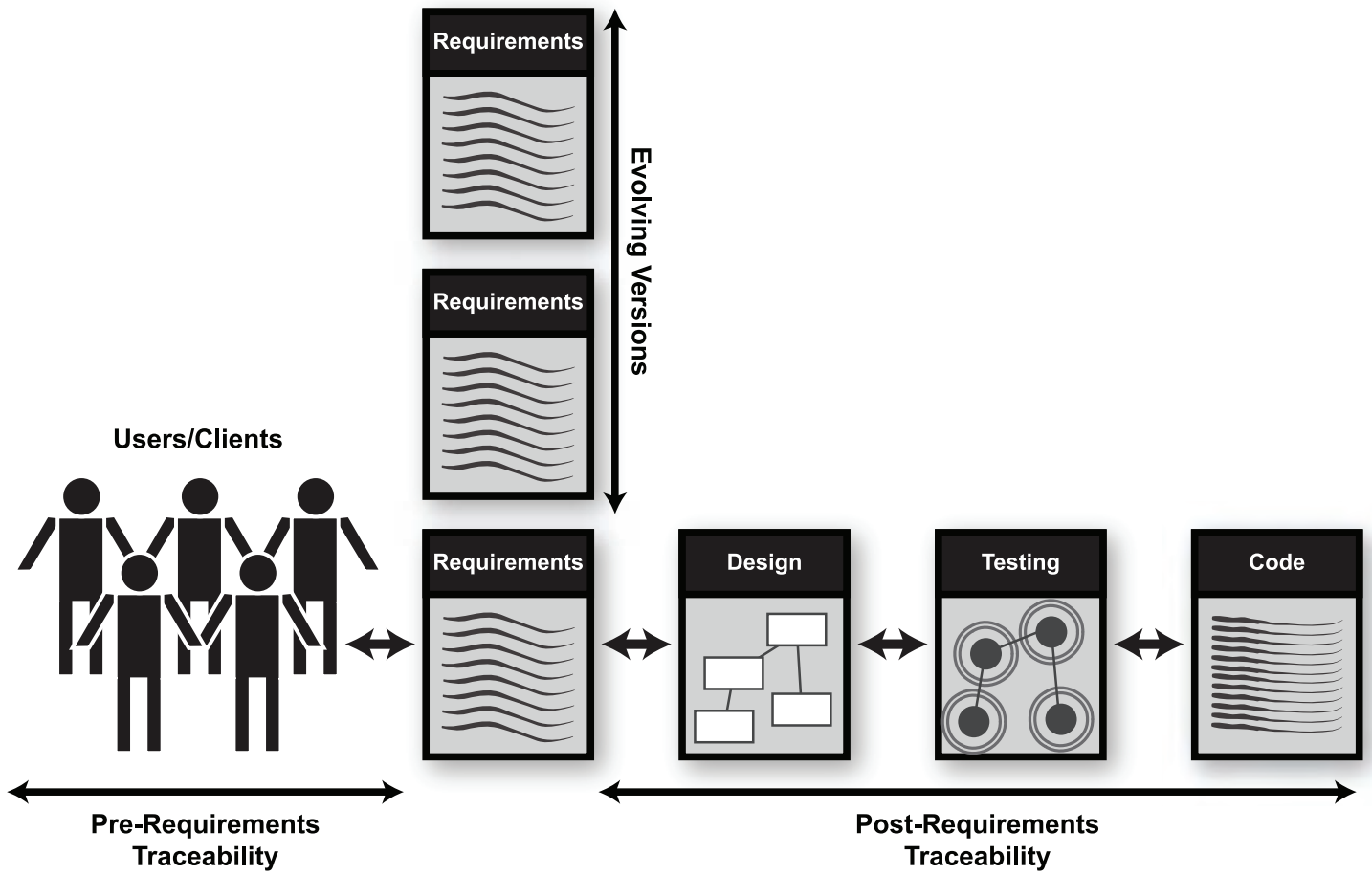
Challenges in Requirement Traceability

In spite of the benefits that traceability offers to the software engineering industry, its practice faces many challenges. These challenges can be identified under the areas of cost in terms of time and effort, the difficulty of maintaining traceability through change, different viewpoints on traceability held by various project stakeholders, organizational problems and politics, and poor tool support.

Cost

One major challenge facing the implementation of traceability is simply the costs involved. As a system grows in size and complexity, capturing the requirement traces quickly becomes complex and expensive [7]. Because of this, the budget for a project implementing traceability must be greater than that of a project without it. However, a project implement-

Figure 2: *An Alternative View of Software Requirements Traceability*



Year	Failed Projects	Challenged Projects	Successful Projects
1994	53%	31%	16%
2006	19%	46%	35%

Table 1: *Comparison of the Standish Group's 1994 and 2006 Results*

ing traceability is far less likely to incur major budget overruns because traceability can detect project problems early in the development process when they are easier and cheaper to correct.

One method of dealing with the high cost of traceability is to practice value-based requirement tracing instead of full tracing. Value-based requirement tracing prioritizes all of the requirements in the system, with the amount of time and effort expended on tracing each requirement depending on the priority of that requirement [7]. This can save a significant amount of effort by focusing traceability activities on the most important requirements. However, value-based tracing requires a clear understanding of the importance of each requirement in the system; it may not be an option if full tracing is a requirement of the customer or the development process standards used for the project.

Alternatively, the high costs of traceability can be approached with the attitude that the costs incurred will save much greater costs further along in the development process due to the benefits that traceability offers to software projects. This method does not solve the problem of the high costs of traceability implementation, but it promotes a healthy attitude towards managing costs for the entire duration of a project instead of merely looking at the short term.

Managing Change

Maintaining traceability through changes to the system is another significant challenge. Studies have shown that change can be expected throughout the life cycle of nearly every software project [8, 9]. Whenever such changes occur, it is necessary to update the traceability data to reflect these changes. This requires discipline on the part of those making the change to update the traceability data, which can be costly in terms of time and effort when the changes are extensive. Unfortunately, strong discipline in maintaining the accuracy of traceability is uncommon, leading to a practice of disregarding traceability information in many organizations [10].

Dealing with change and its impact on traceability is a difficult prospect. Some COTS tools offer assistance with identifying the impact of change on existing traceability data; however, a lot of manual time and effort is still required to update the traceability data [11]. Alternatively, training can help users understand the importance of discipline in maintaining traceability data when changes occur. Focusing on the long-term benefits of traceability instead of the short-term costs can help an organization sustain a healthy attitude toward the costs of maintaining traceability data amidst change.

"If an organization has clear traceability policies in place and provides training on how to comply with these policies, it is likely that traceability will be implemented in a thorough manner consistent with policy."

Different Stakeholder Viewpoints

A contributing factor to poor support for traceability may be the fact that many different viewpoints regarding traceability exist, even among different project stakeholders. These different viewpoints exist primarily because current software engineering standards typically require traceability to be implemented but provide little guidance as to why and how it should be performed [5].

Project sponsors and upper management often view traceability as something that needs to be implemented merely to comply with standards [12]. This leads to a desire to spend as little time as possible on traceability because the benefits outside of standards compliance are not well understood. This viewpoint will likely conflict with that of project engineers familiar with the importance of traceability who will want to ensure that the traceability performed is complete and correct. Perhaps the best way to deal with the problem of different stakeholder viewpoints on traceability is to create an organizational policy on traceability to apply uniformly to all projects. Because the standards requiring traceability are vague, organizations have a lot of leeway in getting their own procedures in place for implementing traceability. This can reduce the amount of confusion about traceability and leads to more consistent viewpoints among the stakeholders involved.

Organizational Problems

Organizational problems also provide a significant challenge to the implementation of traceability. Many organizations view traceability as a mandate from sponsors or a tool for standard compliance [12]. Typically, these organizations do not have a commitment to comprehensive traceability practices. This leads to an ad-hoc practice of traceability, where traceability data is created and maintained haphazardly.

Lack of training poses another challenge [2]. Many organizations do not train their employees regarding the importance of traceability and traceability is not emphasized in undergraduate education. This can lead to resentment on the part of those tasked with creating and maintaining traceability information. They may view the added workload as impacting their productivity due to a staff's insufficient understanding of why traceability is important.

Politics can also play a role. Individuals may be concerned that traceability data will be used against them in performance reviews or as a threat to their job security [13]. This issue can arise because the individual who captures a piece of traceability

Table 2: *Example Traceability Matrix*

System Requirement	Software Requirement	Design Element	Code Module	Test Case
005-00150-80#00505	005-00150-85#00112	Airspeed Calculation	calculate_airspeed()	tc_103.doc
005-00150-80#00506	005-00150-85#00234	Airspeed Display	display_airspeed()	tc_125.doc

information is usually not the one who makes use of it later. Those involved with creating and maintaining traceability data may feel that they are helping others to look good while reducing their own productivity.

The easiest way to correct organizational problems related to traceability is through the use of policy and training. If an organization has clear traceability policies in place and provides training on how to comply with these policies, it is likely that traceability will be implemented in a thorough manner consistent with policy [12].

Poor Tool Support

Poor tool support is perhaps the biggest challenge to the implementation of traceability. Even though the International Council on Systems Engineering (INCOSE) has a survey (see [14]) that lists 31 different tools claiming to provide full traceability support, traceability tool penetration throughout the software engineering industry is surprisingly low. Multiple studies have found the level of commercial traceability tool adoption to be around 50 percent throughout industry [15, 16]. The majority of the remaining companies utilize manual methods (such as manually created traceability matrices for implementing traceability), and a small percentage develop their own in-house traceability tools.

Problems With Manual Traceability Methods

Traceability information can be captured manually through utilizing techniques such as traceability matrices. A traceability matrix can be defined as "a table that illustrates logical links between individual functional requirements and other system artifacts" [8]. Since traceability matrices are in tabular form, they are typically created using a spreadsheet or a word processing application's table function and are independent of the artifacts from which they've captured traceability information. An example traceability matrix is shown in Table 2.

Unfortunately, manual traceability methods are not suitable for the needs of the software engineering industry. In [17], the authors found that the number of traceability links that need to be captured grows exponentially with the size and complexity of the software system. This means that manually capturing traceability data for large software projects requires an extreme amount of time and effort.

Manual traceability methods are also vulnerable to changes in the system. If changes occur to any elements captured in the traceability data, the affected portions of the traceability data must be updated manually. This requires discipline and a significant amount of time and effort spent on link-checking throughout the traceability data. Because of this, it is easy for manually created traceability data to become out-of-sync with the current set of requirements, design, code, and test cases.

Manual traceability methods are also prone to errors that are not easy to catch. Errors can arise from simple typographic mistakes, from inadvertently overlooking a portion of the traceability data (such as an individual requirement), or from carelessness by the individual capturing the data. Because traceability artifacts for large projects are often hundreds or even thousands of pages in length, such errors are difficult to detect when depending on manual methods for error checking.

" ... the number of traceability links that need to be captured grows exponentially with the size and complexity of the software system. This means that manually capturing traceability data for large software projects requires an extreme amount of time and effort."

Because of these disadvantages, manual traceability methods are not suitable for anything other than small software projects. Ralph R. Young stated: "in my judgment, an automated requirements tool is required for any project except tiny ones" [18]. Similarly, Balasubramaniam Ramesh (in [12]) found that traceability is error-prone, time-consuming, and impossible to maintain without the use of automated tools. Then why would nearly 50 percent of software companies use manual traceability methods? Is it because they are all developing tiny projects? This is highly unlikely. In 1994, Orlena Gotel and Anthony Finkelstein [2] found that manual traceability methods were preferred in the industry due to shortcomings in available traceability tools. It is apparent that this problem still exists today because manual traceability methods are still preferred by a significant percentage of software organizations.

Problems With COTS Traceability Tools

Regrettably, currently existing COTS traceability tools are not adequate for the needs of the software engineering industry. Studies have shown that existing commercial traceability tools provide only simplistic support for traceability [5]. Surprisingly, the tools that are available do not fully automate the entire traceability process; instead, they require users to manually update many aspects of the traceability data. This has led some researchers to conclude that poor tool support is the root cause for the lack of traceability implementation [19].

COTS tools are typically marketed as complete requirements management packages, meaning that traceability is only one added feature. The traceability features usually only work if the project methodology is based around the tool itself. Unless the project is developed from the ground up using a particular tool, the tool is unable to provide much benefit without significant rework. Support for heterogeneous computing environments is also lacking.

Although most tools support the identification of impacted artifacts when changes occur, they typically do not provide assistance with updating the traceability links or ensuring that the links and affected artifacts are updated in a timely manner [17]. This means that even when tools are used, the traceability information is not always maintained, nor can it always be trusted to be up-to-date and accurate. This problem is exacerbated by the fact that tools typically only allow primitive actions to be taken (in regards to traceability).

Another issue with tools is that they often suffer problems with poor integration and inflexibility. This has led at least one researcher to conclude that existing traceability tools have been developed mostly for research purposes, and that many projects are still waiting for tools that do not require a particular development or testing methodology [15].

Cost is another major disadvantage. Although the licensing fees vary per tool,

Coming Events

August 24-28
13th International Software Product Line Conference
San Francisco, CA
www.sei.cmu.edu/activities/splc2009

August 31-September 4
17th IEEE International Requirements Engineering Conference
Atlanta, GA
www.re09.org

September 8-10
2009 Unique Identification Forum
Orlando, FL
www.uidforum.com

September 21-24
4th Annual Team Software Process Symposium
New Orleans, LA
www.sei.cmu.edu/tsp/symposium

October 4-9
ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems
Denver, CO
www.modelsconference.org

October 18-21
MILCOM 2009
Boston, MA
www.milcom.org

October 19-23
International Conference on Software Process Improvement 2009
Washington, D.C.
www.icspi.com

2010
22nd Annual Systems and Software Technology Conference

www.sstc-online.org

Coming Events: Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.

the price tends to be thousands of dollars up front, per license, in addition to yearly maintenance fees. Because of this, the cost of using COTS tools is often prohibitive, even for fairly small teams. Such tools are also decoupled from the development environment, meaning that important traceability information—such as code modules that implement requirements—may not be available [20]. For this reason, Ramesh concluded that COTS tools have "very limited utility in capturing dynamic traceability information" [12].

Few solutions are available for the problem of poor tool support for traceability. Many organizations shun COTS tools altogether due to their high cost and inflexibility, instead making use of manual methods such as traceability matrices. Another approach—common among organizations concerned with high-quality traceability information—is to develop elaborate in-house tools and utilities to implement traceability [5]. Unfortunately, this approach is not always feasible since many organizations do not have the manpower or the knowledge necessary to develop such tools. Therefore, poor tool support for traceability currently remains an open problem.

Conclusions

This article has presented an introduction to the benefits offered by traceability and the challenges faced by the practice of traceability in software projects today. Traceability offers benefits to organizations in the areas of project management, process visibility, V&V, and maintenance. Traceability needs to be hardcoded into a process to be replicated iteratively on each and every project. Unfortunately, many organizations struggle to understand the importance of traceability, meaning that these benefits often go unrealized.

In spite of the benefits offered by traceability, its implementation still faces many challenges, especially in the areas of cost, change management, organizational problems, and poor tool support. The lack of quality COTS traceability tools is a significant challenge facing the implementation of traceability in the software engineering industry today. These challenges lead many organizations to implement only as much traceability as is required by their customers.

The challenges faced by traceability are not new. Many of these challenges can be mitigated by process and organi-

zational changes by groups interested in improving their traceability practices. Poor tool support for traceability remains an exception; this is an area that is still an open problem. Existing tools are costly and provide only partial traceability support. This means that implementing traceability is often tedious, requiring a large amount of manual effort.

The lack of quality tools for implementing traceability is not a technically insurmountable problem. The solution simply involves creating cost-effective traceability tools that improve upon the design and feature set of currently available tools. Such tools would serve to greatly improve the practice of traceability in the software engineering industry.◆

References

1. The Standish Group. The Chaos Report. 1994 <www.ibv.liu.se/content/1/c6/04/12/28/The%20CHAOS%20Report.pdf>.
2. Gotel, Orlena, and Anthony Finkelstein. An Analysis of the Requirements Traceability Problem. Proc. of the First International Conference on Requirements Engineering. Colorado Springs, 1994: 94-101.
3. Dömges, Ralf, and Klaus Pohl. "Adapting Traceability Environments to Project Specific Needs." Communications of the ACM 41.12 (2008): 55-62.
4. The Standish Group. The Chaos Report. 2006.
5. Ramesh, Balasubramaniam, and Matthias Jarke. "Toward Reference Models for Requirements Traceability." IEEE Transactions on Software Engineering 27.1 (2001): 58-93.
6. Palmer, J.D. "Traceability." Software Requirements Engineering. Richard H. Thayer and Merlin Dorfman, eds. New York: IEEE Computer Society Press, 1997.
7. Heindl, Matthias, and Stefan Biffl. A Case Study on Value-Based Requirements Tracing. Proc. of the 10th European Software Engineering Conference. Lisbon, Portugal, 2005: 60-69.
8. Wiegers, Karl. Software Requirements. 2nd ed. Redmond, WA: Microsoft Press, 2003.
9. Boehm, Barry. "Value Based Software Engineering." ACM SIGSOFT Software Engineering Notes 28.2 (2003).
10. Clarke, Siobhán, et al. Subject-Oriented Design: Towards Improved Alignment of Requirements, Design, and Code. Proc. of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. Dallas, TX: 325-329.
11. Cleland-Huang, Jane, Carl K. Chang, and Yujia Ge. Supporting Event Based Traceability Through High-Level Recognition of Change Events. Proc. of the 26th Annual International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment. Oxford, England, 2002: 595-602.
12. Ramesh, Balasubramaniam. "Factors Influencing Requirements Traceability Practice." Communications of the ACM 41.12 (1998): 37-44.
13. Jarke, Matthias. "Requirements Tracing." Communications of the ACM 41.12 (1998): 32-36.
14. INCOSE. "INCOSE Requirements Management Tools Survey." 2008 <www.paper-review.com/tools/rms/read.php>.
15. Gills, Martins. "Software Testing and Traceability." University of Latvia. 2005 <http://www3.acadlib.lv/greydoc/Gilla_disertacija/MGills_ang.doc>.
16. Lempia, David L., and Steven P. Miller. Requirements Engineering Management. Proc. of the National Software and Complex Electronic Hardware Standardization Conference. Atlanta, 2006.
17. Cleland-Huang, Jane, Carl K. Chang, and Mark J. Christensen. "Event-Based Traceability for Managing Evolutionary Change." IEEE Transactions on Software Engineering 29.9 (2003): 796-810.
18. Young, Ralph R. "Twelve Requirement Basics for Project Success." CROSSTALK Dec. 2006.
19. Spanoudakis, George, et al. "Rule-Based Generation of Requirements Traceability Relations." Journal of Systems and Software 72.2 (2004): 105-127.
20. Naslavsky, Leila, et al. Using Scenarios to Support Traceability. Proc. of the Third International Workshop on Traceability in Emerging Forms of Software Engineering. Long Beach, CA, 2005: 25-30.

About the Authors

Andrew Kannenberg is a software engineer at Garmin International in Olathe, Kansas, and is currently working on his doctorate in computer engineering at the University of Kansas. He received a bachelor's degree in computer science from the South Dakota School of Mines and Technology and his master's degree in computer science from the University of Kansas.

Garmin International, Inc.
1200 E 151st ST
Olathe, KS 66062
Phone: (913) 397-8200
E-mail: andy.kannenberg@gmail.com

Hossein Saiedian, Ph.D., is currently a professor of software engineering at the University of Kansas. Saiedian's primary area of research is software engineering and in particular technical and managerial models for quality software development. His past research has been supported by the National Science Foundation as well as regional organizations. He has more than 100 publications on a variety of topics in software engineering and computer science and is a senior member of the IEEE. Saiedian received his doctorate in computing and information sciences from Kansas State University.

The University of Kansas
Electrical Engineering and Computer Science
University of Kansas
12600 Quivira RD
Overland Park, KS 66213
Phone: (913) 897-8515
Fax: (913) 897-8682
E-mail: saiedian@ku.edu

21st Annual Systems and Software Technology Conference 2009: Technology: Advancing Precision

Well, there you have it—another successful Systems and Software Technology Conference (SSTC) under our belts. Before turning our attention to 2010, let's look at some of this year's highlights. As we entered the Salt Palace Convention Center after two years away (Las Vegas in 2008, and Tampa in 2007), it felt like coming home. There were the familiar faces, room sets, downtown construction, and wild oddly striped carpet.

The 21st SSTC 2009, themed *Technology: Advancing Precision*, explored how we, as a community, can affect systems and software by fine-tuning technology, thus, advancing the precision of technology.

On the weekend before conference, several engineers attended the International Council on Systems Engineering (INCOSE) Certification course and exam. The course provided an overview of the "INCOSE Systems Engineering Handbook" Version 3.1 and helped engineers prepare for the exam. For those who sacrificed their pre-conference weekend to attend, thank you! Keep us informed on how those exams go.

309 SMXG Director Karl Rogers presents a speaker thank-you gift to Lt. Gen. William L. Shelton after Monday's opening general session.

Monday started with six morning-long tutorials with presentations on safety and security, earned value management, multi-level secure systems, systems of systems engineering, engineering system architectures, and requirements. Monday afternoon started with the official conference kickoff: the Opening General Session with Lt. Gen. William L. Shelton, Chief of Warfighting Integration and Chief Information Officer, Office of the Secretary of the Air Force.

Tuesday began with the presentation of the Stevens Award to Larry Constantine, professor at the University of Madeira (Portugal). Constantine shared several insights, including his belief that the interfaces of applications intended for use must be recognized as more than mere surface manifestations of underlying process and information models, and interaction design must be seen as "more than mere polish of the surface."

The afternoon was dedicated to track presentations with such speakers as Joe Heil and Karen Anderson Cianci.

As is usually the case, the trade show proved to be one of the SSTC highlights. Sponsored by McCabe Software, the trade show hosted about 30 organizations, including our own Software Technology Support Center (STSC). The trade show was almost always packed, but the lines were probably longest during Wednesday's lunch, which featured the attendee-favorite chocolate fountain.

Wednesday's conference proceedings began with two concurrent breakfast speakers: Jorge Edison Lascano, a Fulbright Scholar and Department of Computer Science Faculty Member at the Ecuadorian Army Polytechnic School, and Roger Stewart and Lew Priven, managing directors for the Stewart-Priven Group.

Thursday, the last day of the SSTC, started with six presentation tracks in the topic areas of the developmental life cycle, assurance and security, new concepts and trends, policies and standards, processes and methods, and systems and software modernization.

The closing General Session speaker was Anita D. Carleton, a senior member of the technical staff at the SEI, who gave her presentation "Recognizing Quality Work" during lunch. Along with exploring that the fundamental problem of quality management is not properly recognizing quality work, Carleton shared examples of how the principles of the Team Software Process are expanding into more than just code processes. She also explained how the Mexican government has used these new techniques to achieve their goal of making their country the world leader in providing quality software products and services by 2013.

Every year, we love to see CrossTalk authors as major participants in the proceedings. Along with Stewart and Priven (March/April 2009) and Carleton (May/June 2009), recent CrossTalk contributors speaking at the SSTC included Dr. Benjamin Brosgol (October 2008), Dr. Kelvin Nilsen (February 2009), Dr. David A. Cook (a regular BackTalk contributor), and Arlene F. Minkiewicz (March/April 2009).

CrossTalk is a great forum for information, and the SSTC allows everyone to take that knowledge to another level with learning, conversation, and collaboration. Some of our early feedback includes: "It was a good mix of topics, relevant and timely," "The diversity of topics presented provided for a very good conference," "... lots of good topics and information," "A good mix of general principles and specific techniques and methods," and "I found this year's SSTC to be the most informative so far."

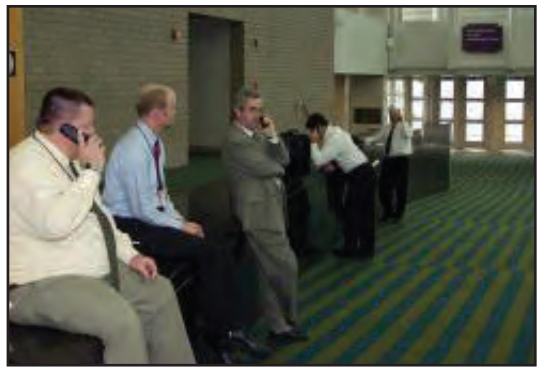
The SSTC, first held in 1989 (known then as the Software Technology Conference), is sponsored by the Air Force through the STSC at Hill AFB, Utah. Every year, the aim has been to provide information and training on software engineering issues and technologies.

For those of you who attended, we extend our gratitude and hope you enjoyed your time at the SSTC and in Salt Lake City. We'll watch for you again next year. SSTC 2010 conference information will begin appearing in your e-mail and mailboxes beginning in mid-August with the "Call for Speakers" brochure. We can't wait to start reading those abstracts!

—CrossTalk and STSC Staff

Photos courtesy of Bob Smith and STSC staff.

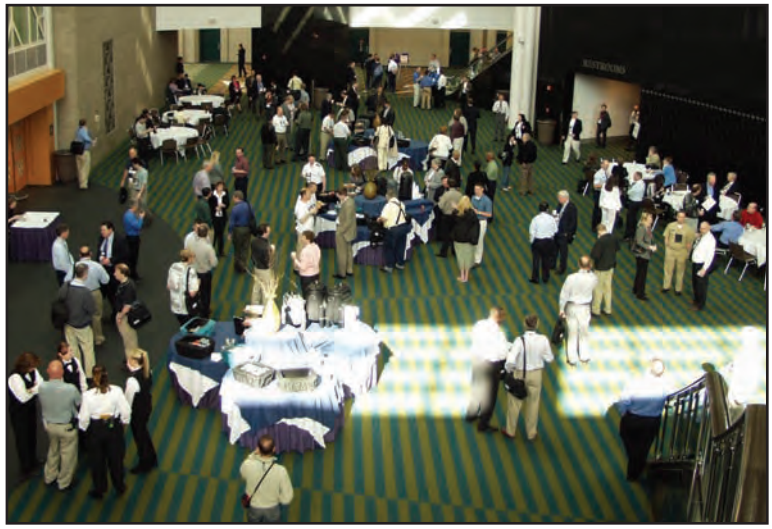
Left: Joe Thiessens from the Software Center of Excellence talks about requirements in one of the conference sessions.
Below right: Attendees use the breaks to play catch-up on work and check messages, or just make lunch plans.



Right: The CrossTalk *author lunch provided an opportunity to meet all our wonderful authors who contribute to our journal every year.*

Below left: SSTC registration staff hands out badges and provides information for all the SSTC attendees.

Below right: Conference attendees congregate outside the trade show for a mid-morning coffee break and snack.



Software Project and Process Measurement

Dr. Christof Ebert
Vector Consulting Services

Software measurement is the discipline that ensures that we stay in control and can replicate successful processes. It applies to products (e.g., ensuring performance and quality), processes (e.g., improving efficiency), projects (e.g., delivering committed results), and people (e.g., evolving competence). This article discusses software measurement and provides practical guidance for project and process measurement.

Goals that are measured will be achieved. In almost every area of life, setting goals and monitoring achievement are the foundations for success. Take a relay team in track or swimming. What drives these athletes to continuously improve? There are always competitors to beat, a number 1 ranking to achieve, or a record to break. Many of the same drivers apply to the software industry. Software development is a human activity and, as such, demands that organizations continuously improve their performance in order to stay in business. Software measurement is a primary approach to control and manage projects and processes and to track and improve performance.

The way software measurement is used in an organization determines how much business value that organization actually realizes. Software measurements are used to:

- Understand and communicate.
- Specify and achieve objectives.
- Identify and resolve problems.
- Decide and improve.

Today, improving a business (and its underlying processes) is impossible without continuous measurements and control. For some industries and organizations, this is demanded by laws (such as the Sarbanes-Oxley Act) and liability regulations. For all organizations, it is a simple question of accounting and finance control. The measurement process is an inherent part of a business process (see Figure 1). Therefore, software engineering—as part of the product development business process—also needs controlling and measurement. Software measurements include performance measurement, project control, and process efficiency and effectiveness.

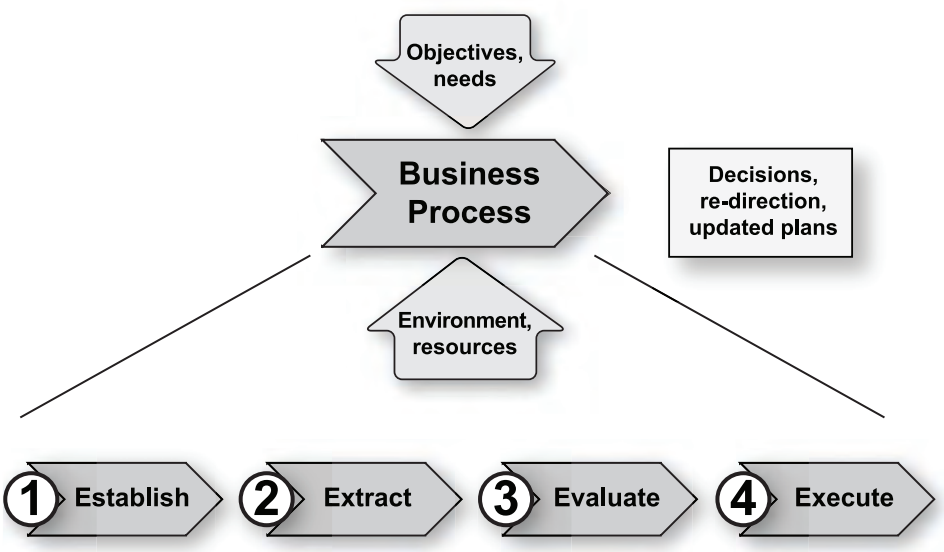
Measurements are management tools. Consequently, measurements must be governed by goals such as reducing project risks or improving test efficiency, otherwise they simply build up into *data cemeteries*. Figure 1 shows this objective-driven generic measurement process, known as E-4 [1]:

1. **Establish** concrete improvement or control objectives and the necessary measurement activities.
2. **Extract** measurements for the established performance control needs.
3. **Evaluate** this information in view of a specific background of actual status and objectives.
4. **Execute** decisions and actions to reduce the differences between status and objectives.

The E-4 measurement process is based on the Deming Cycle (Plan, Do, Check, Act) and extends classic measurement paradigms, such as the goal-question-metric approach, by adding an immediate action-focus. It follows the observation that to effectively and continuously improve a process, it is important to first stabilize it, keep it in its specification limits, and continuously improve it [2]. The major difference is E-4's clear focus on goal-oriented measurement and execution of decisions at the end of the four steps.

The E-4 measurement process is governed by ISO/IEC 15939 [3], a standard that summarizes how to plan and implement a measurement process. It consists of two parts: The first part establishes the measurement program and prepares it within the project or organization. The second part is about execution: You extract data, evaluate it, and execute corrective actions based on the outcome of the measurement analyses. The second part—driven by the first part—sets the standards and defines what to do with the measurements. It is not enough to simply collect numbers and report them. Measurements are only effective if they are embedded as tools to support decision-making and to drive the implementation of decisions. For the fundamentals of software measurement and practical solutions, I refer to the book "Software Measurement" [1].

Figure 1: *A Generic Measurement Process*



Project Measurement

Project measurement is the major application of software measurement. Software measurement is an absolutely necessary precondition for project success, but only one-third of all software engineering companies systematically utilize techniques to measure and control their product releases and development projects [4].

The goal of project measurement is to master the project and finish it according to commitments. What is needed is a way to determine if a project is on track or

not. Whether it is embedded software engineering, the development of a software application, or the introduction of a managed IT-service, demand outstrips the capacity of an organization. As a result, I've observed the acceptance of impossible constraints in time, content, and cost; as a direct consequence of acceptance comes an increase in churn and turmoil—as well as budget overruns, canceled projects, and delays. Still, far too many projects fail to deliver according to initial commitments simply because of insufficient or no adequate measurement.

While many organizations claim that project work is difficult due to changing customer needs and high technology demands, the simple truth is that they do not understand the basics: estimation, planning, and determining progress during the project. Consequently, they fully lose control once customer requirements change.

Project control can help in avoiding these problems by answering a few simple questions derived from the following management activities:

- **Decision-making.** What should I do?
- **Attention directing.** What should I look at?
- **Performance evaluation.** Are we doing a good or bad job?
- **Improvement tracking.** Are we doing better or worse than in the last period?
- **Target setting.** What can we realistically achieve in a given period?
- **Planning.** What is the best way to achieve our targets?

Project monitoring and control is defined as a control activity concerned with identifying, measuring, accumulating, analyzing, and interpreting project information for strategy formulation, planning, and tracking activities, decision-making, and cost accounting. As such, it is the basic method for gaining insight into project performance and is more than just ensuring the overall technical correctness of a project. The most important elements are the existence of a closed loop between the object being controlled, the actual performance measurements, and a comparison of targets against actuals. Figure 2 shows this control loop. The project with its underlying engineering processes delivers results, such as work products. It is influenced and steered by the project goals. Project control captures the observed measurements and risks and relates them to the goals. By analyzing these differences, specific actions can be taken to get back on track or to ensure that the project remains on track. These actions serve as an additional input to the project beyond the original project targets. Making the actions effective is the role of the project manager.

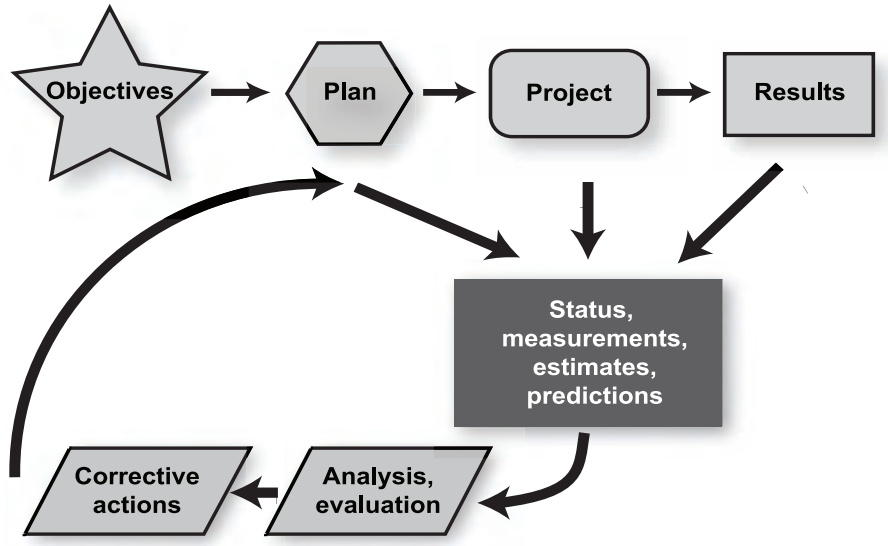


Figure 2: *Measurements Provide a Closed Feedback Loop for Effective Corrective Action*

A client recently asked us for help in improving engineering efficiency at their organically grown embedded software business. Software costs initially did not matter when there were only a few engineers. As things progressed, software dominated hardware and cost increases were out of control. Customers liked the technology but increasingly found quality issues and, when the customers made audits, did not really find a decent engineering process. Our Q&A with the client went like this:

Us: "How do you set up a software project?"

The Client: "Well, we take the requirements, build a team of engineers, and follow our V-shaped master plan."

Us: "Is the planning typically accurate?"

The Client: "No, but we don't have any better basis."

Us: "How do you make sure you have the right quality level?"

The Client: "Well, we do a unit test, integration test, and system test. But, in fact, we test too much and too late."

Us: "Is your cost in line with expectations or could you do better?"

The Client: "We really don't know about how this all is measured and what to learn from measurements. That's why we invited you."

In terms of software and technology, this client was way above average. However, it was unclear to management and engineers how to assess status, mitigate risks, and improve performance. The project managers had no history database to decide on release criteria. Requirements were changing, but without any control. It is exactly this pattern of exciting technology and skills—combined with fast growth but insufficient engineering and management processes—that eventually hits many companies in embedded software development. Finally, the client's customers concluded that despite all of the technical advantages, processes and practices were below current professional software engineering expectations. A natural first step on our side was to introduce a lean yet effective measurement program.

To get started without much overhead, our team recommended a lean set of project indicators [1, 5, 6]. They simplified the selection by reducing the focus on project tracking, contractor oversight, and program management perspective. Here is our short list of absolutely necessary project measurements:

- **Requirements status and volatility.** Requirements status is a basic ingredient to tracking progress based on externally perceived value. Always remember that you are paid for implementing requirements, not just generating code.
- **Product size and complexity.** Size can be measured as either functional size in function points or code size in lines of code or statements. Be prepared to distinguish according to your measurement goals for code size between what is new and what is reused or automatically generated code.
- **Effort.** This is a basic monitoring parameter to assure that you stay within budget. Effort is estimated up-front for the project and its activities. Afterwards, these effort elements are tracked.
- **Schedule and time.** The next basic

monitoring measurement is ensuring that you can keep the scheduled delivery time. Similar to effort, time is broken down into increments or phases that are tracked based on what is delivered so far. Note that milestone completion must be lined up with defined quality criteria to avoid poor quality being detected too late.

- **Project progress.** This is the key measurement during the entire project execution. Progress has many facets: Simply look to deliverables and how they contribute to achieving a project's goals. Typically, there are milestones for the big steps, and earned value and increments for the day-to-day operational tracking. Earned value techniques evaluate how results (such as implemented and tested requirements or closed work packages) relate to the effort spent and elapsed time. This then allows for estimating the cost to complete and remaining time to complete the project.
- **Quality.** This is the most difficult measurement, as it is hardly possible to accurately forecast whether the product has already achieved the right quality level that is expected for operational usage. Quality measurements need to predict quality levels and track how many defects are found compared to estimated defects. Reviews, unit test, and test progress and coverage are the key measurements to indicate quality. Reliability models are established to forecast how many defects still need to be found. Note that quality attributes are not only functional but also relate to performance, security, safety, and maintainability.

These measurements must be actively used for the weekly tracking of status, risks, and progress (e.g., increment availability, requirements progress, code delivery, defect detection), while others are used to build up a history database (e.g., size, effort). Most of these measurements, such as defect tracking, are actually by-products from operational databases. This ensures sufficient data quality to compare project status across all projects of a portfolio. External benchmarks (such as provided in [1,6]) help in bootstrapping a measurement program as they immediately point towards inefficiency and below-average performance.

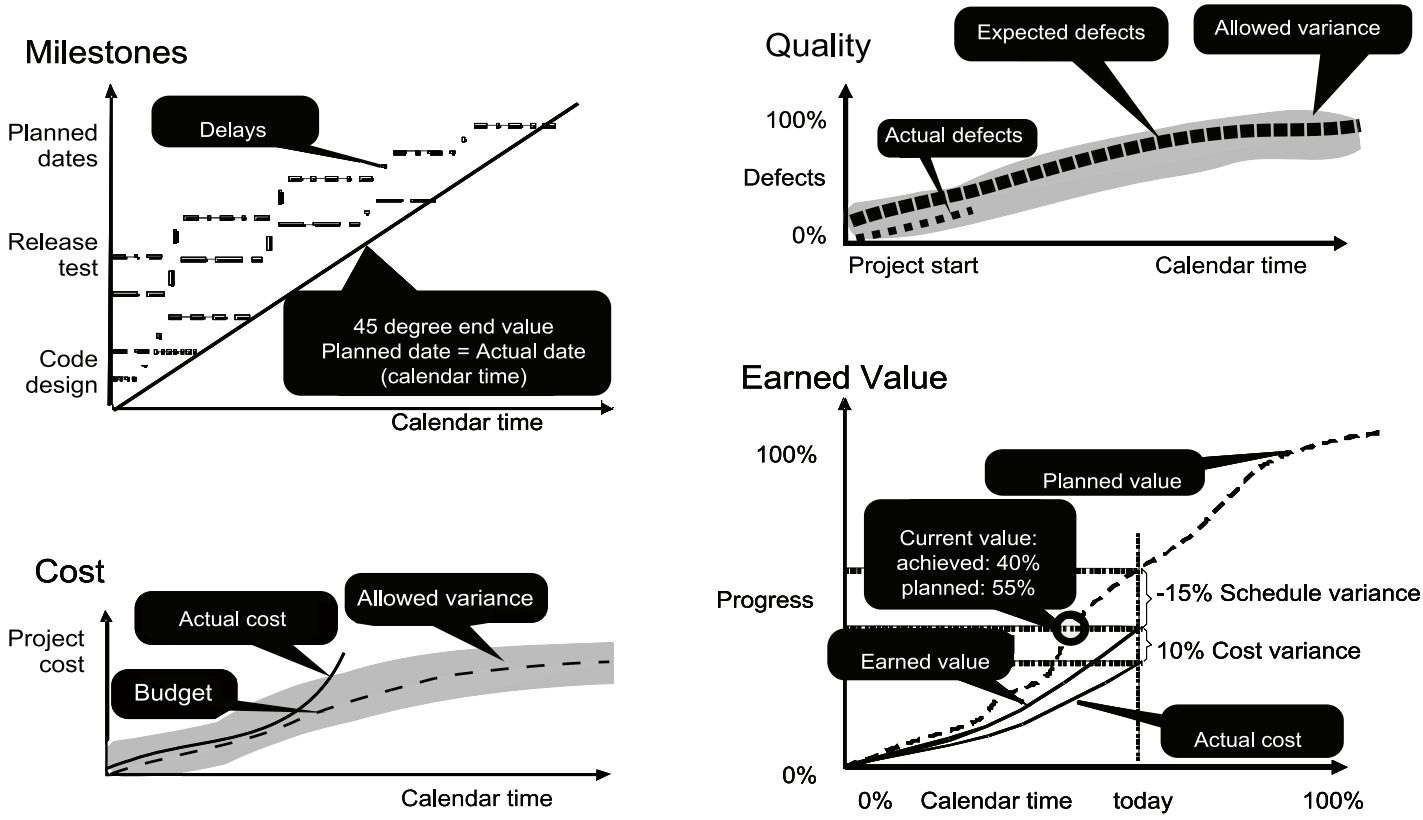
To ease monitoring and avoid getting lost in the fog of numbers, projects should aggregate the relevant information in a standardized dashboard. Figure 3 shows a simplified dashboard of how we have utilized it with many clients. It includes milestone tracking, cost evolution, a selection of process measurements, work product deliveries, and faults with status information. There can be both direct measurements (e.g., cost) as well as indirect measurements and predictions (e.g., cost to complete). Such dashboards provide information in a uniform way across all projects, thus not overloading the user with different representations and semantics to wade through. All projects within the organization must share the same set of consistent measurements presented in a unique dashboard. A lot of time is actually wasted by reinventing spreadsheets and reporting formats when the project team should be focused on creating value.

Measurements—such as schedule and budget adherence, earned value, or quality level—are typical performance indicators that serve as *traffic lights* on the status of the individual project. Only projects in *amber* and *red* status that run out of agreed variance (which, of course, depends on the maturity of the organization) would be investigated. The same dashboard is then allowed to drill down to more detailed measurements to identify root causes of problems. When all projects follow a defined process and utilize the same type of reporting and performance tracking, it is easy to determine status, identify risks, and resolve issues—without getting buried in the details of micromanaging the project.

Process Measurement

Software engineering processes determine the success of organizations as well as how they are perceived in the marketplace.

Figure 3: *Measurement Dashboard Overview*



Purchasing organizations worldwide are using tools such as CMMI to evaluate their suppliers, be it in automotive, information/communication technologies, or governmental projects [1, 7]. Markets have recognized that continuous process improvement contributes substantially to cost reductions and quality improvement. An increasing number of companies are aware of these challenges and are proactively looking at ways to improve their development processes [8]. Suppose a competitor systematically improves productivity at a relatively modest annual rate of 10 percent (as we can see in many software and IT companies). After only three years, the productivity difference is $(1.1^3) = 1.33$, or a 33 percent advantage. This directly translates into more capacity for innovation, higher margins, and improved market positioning.

Software measurement is a necessary precondition to performance improvement. The concept of objective-driven process improvement will focus processes on the objectives they must achieve. Processes are a means to an end and need to be lean, pragmatic, efficient, and effective—or they will ultimately fail, despite all the push one can imagine. Figure 4 shows this goal-driven relationship from business objectives to concrete annual performance objectives (on an operational level) to specific process performance measurements.

What follows are eight integral software measurement steps, showing how objective-driven process improvement is translated into concrete actions, and how software measurements are used to improve processes. Each step includes an example—all taken from the same medium-sized company that Vector recently consulted—showing how we put each step into action.

Step 1: Identify the organization's improvement needs from its business goals

These business goals provide the guidance for setting concrete engineering performance improvement objectives on a short-term basis. Example: The business goal with our client was to improve revenues and cash flows and to reduce cost of non-performance from schedule delays.

Step 2: Define and agree on the organization's key performance indicators (KPIs)

Such KPIs are standardized across the organization and ensure visibility, accountability, and comparability. Naturally, KPIs must relate to the business goals. Measurements should drive informed decision-making. They must be used for effectively communicating status and progress against the objectives set for the business, process, or project. Measurements, both direct and indirect, should be periodically evaluated in conjunction with the driving objectives and to identify problems or derive decisions. Example: The selected performance indicator was schedule predictability, measured as the normalized delays compared to the originally agreed deadline. Schedule changes (after a project had started) were not considered in this measurement. This avoided the argument that a specific delay was justified due to changing requirements. Once such an excuse is accepted, delays typically increase—as do costs (due to the changes). In this case, we found almost unanimous agreement from product managers and business owners who found that the lack of schedule changes helped avoid the downward spiral of requirements changes, delays, and cost overruns.

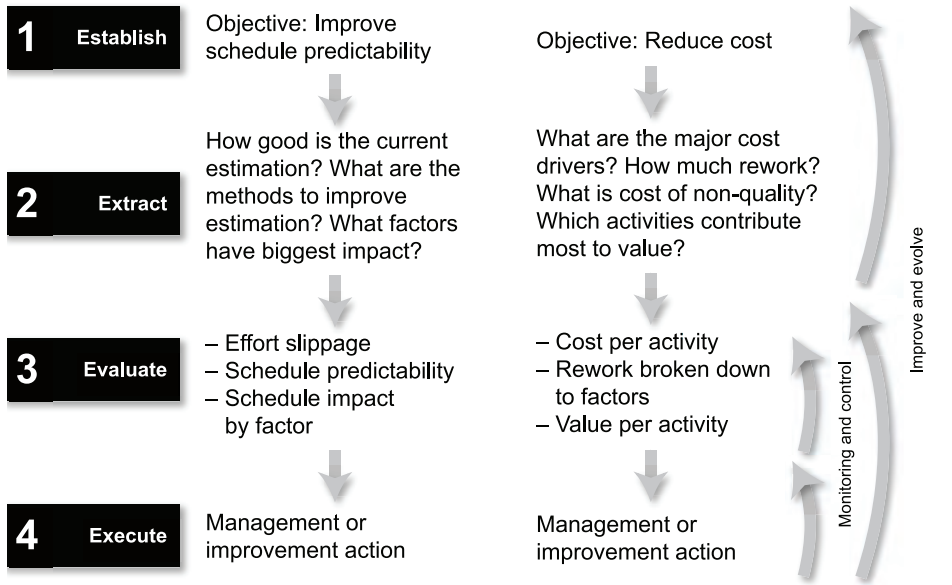


Figure 4: *Applying the E-4 Measurement Process to Achieve Improvements*

Step 3: Identify the organization's hot spots, such as areas where they are feeling the most pain internally or from customers

Typical techniques include root cause analysis of defects and customer surveys. Example: Average schedules in the company were 45 percent behind initial commitments. A root cause analysis of delays was performed: 40 percent of delays came from insufficient project management; 30 percent from changing requirements; 20 percent from supplier delays; and 10 percent from other causes. Looking to the highest-ranking root causes, we found insufficient planning and control within the project. Often, requirements were added or changed following a customer question—rather than including the customer in the trade-off and impact analysis. When speaking to client-customers, this became even more evident: Customers perceived such ad-hoc changes as a weakness and missed clear guidance and leadership on feature content.

Step 4: Commit to concrete improvement objectives

These improvement objectives should directly address the mentioned weaknesses and simultaneously support the overarching business objective. We use the acronym SMART to outline good objectives:

- **S**pecific (precise).
- **M**easurable (tangible).
- **A**ccountable (in-line with individual responsibilities).
- **R**ealistic (achievable).
- **T**imely (suitable for current needs).

These objectives are reviewed and approved by upper-level managers before proceeding further. This ensures that priorities are appropriate and that nothing relevant has been overlooked or misunderstood. Example: Two improvement objectives, improving estimations and improving requirements development, were agreed upon. The respective performance targets were agreed to in a management seminar to achieve buy-in. Each project was required to have two estimates where the first was allowed to deviate by 20 percent and the second to deviate by 10 percent. After the project started, the requirements change rate was required to be under 20 percent—except that customers would, after a clear decision-making process, agree on trade-offs and pay for the changes. Time-boxing and incre-

mental development with prioritized requirements was then introduced to achieve improvement objectives. Requirements priorities were agreed upon across impacted stakeholders from product management, engineering, and marketing/sales before each new project started.

Step 5: Identify specific levers[1] to start improvements and connect them to ROI planning

This is typically best done when using a process improvement framework such as CMMI [6]. This framework provides the necessary guidance regarding which best practices to apply and how processes relate to each other. Without the right levers, chances are high that objectives will not be reached. Example: Focus was on requirements development, requirements management, technical solutions, project planning, and project monitoring and control. Project managers were educated in project management techniques and negotiation skills.

Step 6: Perform a brief gap analysis of the selected process areas to identify strengths and weaknesses

This systematic look at weaknesses helps to focus limited engineering resources where it matters most. Example: Requirements were collected rather than developed; requirements management satisfied the basic need for change management; engineering was too technology-driven and was extended to capture business reasoning; project planning showed severe weaknesses in estimation and feasibility analysis; and project monitoring and control showed weaknesses in getting stakeholder agreement on changes.

Step 7: Develop a concrete action plan for the identified weaknesses

Avoid trying to change all weaknesses at the same time. Use increments to subdivide bigger changes. Consider available resources and skills and get external support if you lack competencies, such as change management. Example: The most urgent need was project planning. A dedicated one-month initiative was launched right away to install a suitable estimation method and to train people on it. A tool for feasibility analysis was introduced in parallel because not much of the organization's own historical data was available. A historical database was installed for a set of key project measurements. In a second phase, earned value analysis was introduced. After three months, requirements development was launched under the leadership of product management.

Step 8: Implement improvements and deliver tangible results

Implement the agreed-upon changes to operational projects and measure progress against the committed improvement objectives. Example: Performance measurements were collected from all ongoing projects. There weren't reprimands for insufficient performance, but it was carefully analyzed. We found with this client that too many requirements changes lacked a clear specification, analysis, and commitment by the product manager. Consequently, a strong focus was given towards change management and change review boards[2]. A weekly project review was introduced after a few weeks, enhanced with daily Scrum meetings of development teams. Requirements changes passing the change review board had to have their proper business case or were not accepted. Although the results proved valid, marketing and sales were unhappy because they wanted flexibility and no accountability for the changes. This improvement reduced requirements changes (within the first three months) substantially. The first few projects had 20 percent schedule overrun (compared to the previous average of 45 percent) and two of them were even close to 10 percent. These two were further evaluated to identify best practices.

As it turned out, the project managers demanded requirements reviews by product managers and testers before accepting requirements to change review boards. The quality of requirements substantially improved. This change was immediately pushed forward by senior management for all projects. As it turns out, testers were unhappy because they didn't have the time scheduled for doing the reviews. Rather than demanding overtime, senior management asked for *five percent slots* (two hours) each week; this proved to be sufficient time to review one to three requirements with the necessary depth.

Getting More From Your Measurements

Measurement programs mostly fail because they are disconnected from actual business. Too often, things are measured that do not matter at all and there is usually no clear improvement objective behind what is measured. Often the collected measurements and resulting reports are useless, only creating additional overhead. In the worst cases, they hide useful and necessary information. We have seen reports on software programs with more than 50 pages full of graphs, tables, and numbers. When asked about topics such as cost to complete, expected cost of non-quality after handover, or time-to-profit, the organization did not have a single number. Sometimes numbers are even created to hide reality and attract attention to what looks good. Be aware that measurements are sometimes abused to obscure and confuse reality.

The primary question with software measurement is not "What measurements should I use?" but rather "What do I need to improve?" It is not about having many numbers but rather about having access to the exact information needed to understand, manage, and improve your business. This holds true for both project and process measurement.

As a professional in today's fast-paced and ever-changing business environment, you need to understand how to manage projects on the basis of measurements and forecasts. You need to know which measurements are important and how to use them effectively. Here are some success factors to pragmatically use measurements:

- Estimate project time and effort and realistically set deadlines.
- Check feasibility on the basis of given requirements, needs, and past project performance (productivity, quality, schedule, and so on). Do not routinely overcommit.
- Manage your requirements and keep track of changes. Measure requirements changes and set thresholds of what is allowed.
- Know what *value* means to your client or customer. Track the earned value of your project.
- Understand what is causing delays and defects. Do not let delays accumulate. Remove the root causes and do not simply treat the symptoms.
- Be decisive and communicate with a fact-based approach. Avoid disputes with management, clients, and users.
- Deliver what matters. Use measurements to keep commitments.
- Continuously improve by analyzing your measurements and then implementing respective objective-driven changes. Follow through until results are delivered.

Practitioners should help management so that they make decisions on the basis of measurements. This will give management the necessary information

before and after the decision so that they can follow the effects and compare them to the goals. Managers should ensure that decisions are made based on facts and analyses. They should always consider number 4 in the E-4 process (executing decisions and actions), and make sure that decisions move the organization towards agreed-upon objectives.

Accountability means setting realistic and measurable objectives. Objectives such as *reduce errors* or *improve quality by 50 percent* are not useful. The objective should be clear and tangible: *Reduce the number of late projects by 50 percent for this year compared to the previous year.* These objectives must end up in a manager's key performance indicators. Projects using unclear oral, memo, or slide reports will most certainly fail to deliver according to commitments. Projects using a standard spreadsheet-based progress reporting—related to requirements, test cases, and defects versus plans—will immediately receive the necessary attention if they deviate. With this attention, corrective action will follow, and the project has a good chance to recuperate because of sufficiently early timing.

Goals that are measured will be achieved. The reason is very simple: Once you set a SMART objective and follow it up, you make a commitment to both yourself and your team or management group. As humans—and specifically as engineers—we want to deliver results. Measurement provides the stimulus and direction to reach our goals.◆

References

1. Ebert, Christof, and Reiner Dumke. Software Measurement. New York/Heidelberg: Springer-Verlag, 2007.
2. Juran, Joseph M., and A. Blanton Godfrey. Juran's Quality Handbook. New York: McGraw-Hill Professional, 2000.
3. ISO/IEC. ISO/IEC 15939:2002. Software Engineering – Software Measurement Process. 2002.
4. Kraft, Theresa A. "Systematic and Holistic IT Project Management Approach for Commercial Software With Case Studies." Information System Education Journal 63.6: 1-15. 23 Dec. 2008 <http://isedj.org/6/63/>.
5. Carleton, Anita D., et al. "Software Measurement for DoD Systems: Recommendations for Initial Core Measures." Technical Report CMU/SEI-92-TR-19. Sept. 1992 <www.sei.cmu.edu/pub/documents/92.reports/pdf/tr19.92.pdf>.
6. Ebert, Christof, and Capers Jones. "Embedded Software: Facts, Figures and Future." IEEE Computer 42.4: 42-52, Apr. 2009.
7. Chrissis, Mary Beth, Mike Konrad, and Sandy Shrum. CMMI: Guidelines for Process Integration and Product Improvement. 2nd ed. Boston: Addison-Wesley Professional, 2006.
8. Gibson, Diane L., Dennis R. Goldenson, and Keith Kost. "Performance Results of CMMI-Based Process Improvement." Technical Report CMU/SEI-2006-TR-004. Aug. 2006 <www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr004.pdf>.

Notes

1. "Levers" in this case mean to set up the improvement project in a way that the different changes follow some order, won't come ad-hoc and isolated, and would thus meet objectives.
2. Change review boards are staffed with engineering and product managers and ensure that both technical and business rationale and impacts of changes are considered. They then make informed and firm decisions.

About the Author

Christof Ebert, Ph.D., is managing director and partner at Vector Consulting Services. He is helping clients worldwide to improve technical product development and to manage organizational changes. Prior to working at Vector, he held engineering and management positions for more than a decade in telecommunication, IT, aerospace, and transportation. A measurement practitioner who has worked for *Fortune* 500 companies, Ebert authored "Software Measurement," published by Springer, now in its third fully revised edition.

Vector Consulting Services
Ingersheimer Straße 24
D-70499 Stuttgart
Germany
Phone: +49-711-80670-0
E-mail: christof.ebert@vector-consulting.de

Be a CrossTalk Backer

CrossTalk would like to thank the accompanying organizations, designated as CrossTalk Backers, that help make this issue possible.

CrossTalk Backers are government organizations that provide support to forward the mission of CrossTalk. Co-Sponsors and Backers are our lifeblood.

Backer benefits include:

- An invaluable opportunity to share information from your organization's perspective with the software defense industry.
- Dedicated space in each issue.
- Advertisements ranging from a full to a quarter page.
- Web recognition and a link to your organization's page via CrossTalk's Web site.

Please contact Kasey Thompson at (801) 586-1037 to find out more about becoming a CrossTalk Backer.

CrossTalk would like to thank our current Backers:

309th Software Maintenance Group

Cost Analysis Group

OO-ALC Engineering Directorate

309th Electronics Maintenance Group

A Perspective on Emerging Industry SOA Best Practices©

Larry Pizette, Salim Semy, Geoffrey Raines, and Steve Foote
The MITRE Corporation

Using a service-oriented architecture (SOA) approach allows organizations to become both more efficient in meeting their current needs and more agile in meeting future—and possibly unknown—challenges. SOA, however, is not a panacea. As with any large-scale systems integration effort, there are challenges with employing SOA techniques effectively. This article presents industry best practices to deal with key SOA challenges.

Given the anticipated benefit of delivering business and operational value improvements—such as cost savings, better business processes, and increased accessibility to information—SOA has become a high-priority focus area for the federal government. As various organizations in the government research adopting an SOA, they often struggle with fundamental questions:

- **Why should we adopt an SOA approach for our IT portfolio?** Is it the best approach for our organization?
- **What are the inhibitors to an SOA approach?** What causes the failure of SOA initiatives and how do we avoid these pitfalls?
- **How do we reduce risk?** What are the approaches for reducing risk, given various implementation techniques, standards, and commercial products?
- **What results should we expect from implementing an SOA?** How should the organization govern its SOA to maximize the business value of its investment?

As the government evolves their architectures to a service orientation on a large scale, leadership will look for the lessons learned by industry and other government organizations. While practical experiences clearly demonstrating the benefits of an SOA approach will continue to emerge, research into initial experiences suggests that the following set of eight best practices can enable an effective SOA adoption strategy.

1. Determine if an SOA is the best approach

When used appropriately, an SOA approach can provide significant value to an organization—but it is not always the right approach or the best fit. In many situations, underlying business requirements make the adoption of contemporary SOA technologies impractical. For example, specialized security requirements, an organization's inherent network limitations, or high bandwidth data feeds can be impediments to applying industry-standard contemporary SOA technologies.

It is possible to employ custom technologies to meet specialized needs; however, there is significant value in using industry standards when establishing an SOA portfolio. Implementations that need to deviate from standards are less likely to benefit from interoperability and later improvements in standards-based technology.

> *"As the government evolves their architectures to a service orientation on a large scale, leadership will look for the lessons learned by industry and other government organizations."*

Before embarking on an SOA implementation, consider whether your underlying business and technical requirements can be met with contemporary SOA technologies and standards.

2. Start SOA activities with the focus of solving business and operational challenges

Federal leaders should employ services to support key business processes and SOA efforts should be driven by the organization's business and operational goals. Randy Heffner explains that an SOA is always best with a business process focus:

> You can build a much stronger conversation around doing SOA on a project if you focus conversations on the business' high-priority process pain points and opportunities. [1]

Contemporary SOA approaches facilitate a reduction in an organization's IT portfolio by providing services that are commonly used across many business processes. These business process services enable an organization's needs to be reliably met by another organization's capabilities. Gene Leganza writes:

> As government agencies re-engineer their business processes to provide horizontal integration to improve services to citizens, other government agencies, and their private-sector partners, SOA allows the agencies to design application components that instantiate the atomic elements of business service delivery in explicit pieces. [2]

When used properly, an SOA enables IT to support business goals. By focusing on services that provide business process capabilities reused across the enterprise, it is more likely that the enterprise will realize the value of the SOA investment.

3. Examine your data, realizing that an SOA does not solve data problems—and it may expose them [3]

The flexibility of SOA in decoupling applications from data may expose issues with semantic differences in data, data quality, and ensuring data availability.

Fundamentally, services share data and, unless providers and consumers have a common understanding of the data that constitutes the payload of a service, shared services will not be possible. When deploying an SOA, it is important to consider your data by defining a common data or abstraction layer, developing mappings

between internal schemas and a common vocabulary across the community, and paying special attention to the governance for maintaining data quality.

A proper focus on data will enable interoperability among consumers and providers and lower implementation risks for the enterprise SOA.

4. Start small, learn, and evolve

Employing the *big-bang* approach to SOA adoption is unlikely to be successful due to the inherent risks of very large-scale software development, requirements complexity, and the challenges of establishing a new architecture across large organizations with significant legacy infrastructure and diverse computing needs.

These factors, coupled with the risk of a large deployment, point us towards starting small, learning, and evolving. SOA initiatives should begin by addressing a business problem constrained in scope, focusing on piloting the architecture, ensuring that clearly defined success criteria exist, and capturing the lessons learned to educate the enterprise and improve future SOA implementations. Ron Schmelzer indicates that organizations should start with a small business problem, adding:

> Service-oriented architects must ... maintain a pragmatic mental picture for how the organization can evolve iteratively while still maintaining a single, cohesive vision of the organization's architecture. [4]

Narrowing the initial scope of an SOA implementation to one or two business processes will help keep the SOA at a manageable and realistic size.

Employing a *start small, learn, and evolve* strategy will minimize risk and reduce the time it takes an organization to realize value from its SOA investment.

5. When establishing an SOA, have a long-term vision

The long-term vision for an SOA implementation is frequently organizational agility and reduced cost, allowing an organization to respond to changing needs quickly and utilize IT resources more effectively. These objectives can be realized through service reuse, ease of interoperability, reduced integration and maintenance costs, and the ability to deploy new capabilities quickly.

An initial SOA implementation that is scalable and capable of expanding in scope and requirements will ensure its growth to meet future and unanticipated needs. Heffner recommends to:

> ... craft your SOA strategy so that investments are made: 1) in line with work done and business value delivered today on each business technology solution delivery project, and 2) as investments across a portfolio of projects, maintaining a significant trajectory ... toward your long-term goals. [1]

SOA implementations should be designed with the expectation that requirements will evolve and should be built to allow for scalability and new capabilities.

6. Governance is a key component of the SOA

SOA technologies can be applied to individual projects, but the changes necessary for an enterprise-wide adoption can only be achieved by putting the right policies and processes in place to bridge the enterprise architecture with the business strategy.

Governance is an essential element of an SOA; it is the vehicle for creating, communicating, and enforcing SOA policies, roles, and responsibilities across the enterprise. The Organization for the Advancement of Structured Information Standards (OASIS) states that:

> SOA governance should be considered an extension of existing IT governance that deals with the decision rights, processes, and policies that are put into place to encourage the adoption and operation of a SOA that may cross ownership boundaries. [5]

Example process areas that should be governed include service life cycle, service versioning, service monitoring, service registries, and service testing [6].

Governance is necessary for establishing trust so that consumers and producers have a set of established expectations for IT services essential to meeting their business needs.

7. Integrate security throughout the SOA life cycle

A primary objective of applying service orientation to a system's architecture is to facilitate broader user access to information stored within that system. A challenge is enabling information sharing while protecting and securing the information being shared [7].

This security challenge can be successfully conquered by dividing it into three major areas and systematically tackling each one: empowering unanticipated users (if an SOA will be used to implement an information-sharing strategy, which requires access privileges for unanticipated users), establishing trust across organizational boundaries, and mitigating newly exposed vulnerabilities. Federal leaders and security architects may need to establish enterprise-wide authentication and authorization mechanisms in order to support access by unanticipated users. Attribute-based access control and other modern security techniques can be leveraged to provide this capability.

The successful implementation of an SOA requires that the right security mechanisms are applied to the right services. Also, security should be balanced with other considerations, such as performance and scalability.

8. Set realistic ROI expectations during SOA implementation

One motivation for moving to an SOA is the promise of cost reduction in operations, reuse, and future systems integration.

While cost savings can be a realistic expectation, an organization should expect upfront costs when implementing SOA techniques for the first time. Causes of upfront costs can be the learning curve associated with modifying legacy applications to create service offerings, the lack of technical staff familiarity with the technologies, and the need for new infrastructure. Infrastructure costs may include middleware (e.g., an enterprise service bus), security components, *quality of service* monitoring software, and hardware and network upgrades. James Kobielus articulates the investment consideration well:

> The upside of SOA is that the marginal cost of building new applications will continue to drop as the service-reuse rate climbs. The catch is that there's a significant ramp-up cost, because adopting an SOA means you're going to need to rethink many of your traditional approaches to application modeling, development, integration, deployment and management. [8]

Cost savings may occur at the enterprise-level eventually, but not necessarily at the project level.

An organization adopting SOA approaches needs to have a realistic expectation on how much investment is needed and the expected ROI.

Conclusion

The ability to leverage IT resources across the network—to adapt to evolving requirements and to rapidly deliver new functionalities to meet users' needs—is at the core of a networked enterprise.

SOA practices can help realize this vision by establishing shared and composable services. For example, DoD Chief Information Officer John Grimes stated:

> One of our challenges, which is true for most large organizations—both public and private—has been the transition from an era where local commands built local-area networks and developed local applications for customer requirements to where organizations have to work together for interoperability to get the right information to the right person at the right time around the globe. [9]

The SOA best practices described in this article are intended to serve as a baseline for successful SOA implementations. They illustrate that an SOA is more than a group of systems or purely a software architecture; SOA changes the character and agility of the underlying IT infrastructure that is available to an organization's senior leadership team and decision-makers. While technology is a key part of employing SOA techniques, other IT management issues—such as changing the organization's culture toward providing and consuming services and implementing effective governance processes to continually align the IT portfolio with business requirements—are equally important.◆

References

1. Heffner, Randy, et al. "Pursuing SOA In Hard Times: Adjusting Best Practices for SOA When Resources Are Tight." Forrester Research. 11 Nov. 2008 <www.forrester.com/Research/Document/0,7211,47474,00.html>.
2. Leganza, Gene, et al. "Why Is SOA Hot in Government?" Forrester Research. 12 Dec. 2006 <www.forrester.com/Research/Document/0,7211,40673,00.html>.
3. Friedman, Ted, and Craig Muzilla. eBiz. Webinars. "Where Data Meets SOA: Data Services." 28 June 2006 <www.ebizq.net/webinars/6953.html>.
4. Schmelzer, Ronald. "Right-Sizing Services." ZapThink. 15 Nov. 2005 <www.zapthink.com/report.html?id=ZAPFLASH-20051115>.
5. OASIS. "IT Governance and SOA Governance." 4 Apr. 2007 <http://wiki.oasis-open.org/soa-rm/TheArchitecture/Governance>.
6. Woolf, Bobby. developerWorks. IBM. "Introduction to SOA Governance." July 2007 <www.ibm.com/developerworks/ibm/library/ar-servgov/>.
7. World Wide Web Consortium. "Web Services Architecture." 11 Feb. 2004 <www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.
8. Kobielus, James. "The ROI of SOA: The more you reuse, the more you save." Network World. 10 Oct. 2005 <www.networkworld.com/techinsider/2005/101005-roi-of-soa.html>.
9. Rosenberg, Barry. "Interview: John Grimes." Defense Systems. 17 Nov. 2008 <http://defensesystems.com/Articles/2008/11/Interview-with-John-Grimes.aspx?Page=1>.

About the Authors

Larry Pizette is a principal software systems engineer for The MITRE Corporation's Command and Control Center, supporting a variety of government sponsors. Previously, he held software engineering management positions of increasing responsibility at firms developing enterprise-scale commercial systems. Pizette has a bachelor's degree in computer science from the University of Massachusetts and an MBA from New York University.

The MITRE Corporation
202 Burlington RD
Bedford, MA 01730-1420
E-mail: lpizette@mitre.org

Geoffrey Raines is a principal software systems engineer for The MITRE Corporation's Command and Control Center, supporting a variety of government sponsors. Previously, he was the vice president and chief technical officer of Electronic Consulting Services, Inc.—an information technology and engineering consulting professional services firm, where he developed engineering solutions for federal clients. He has a bachelor's degree in computer science from George Mason University.

The MITRE Corporation
7525 Colshire DR
McLean, VA 22102-7539
E-mail: soa-list@lists.mitre.org

Salim Semy is a lead software systems engineer for The MITRE Corporation's Command and Control Center. Most recently, he focused on developing a framework and identifying design patterns to deliver service-oriented solutions to constrained networking and computing environments. Semy has a bachelor's degree in software engineering from the University of Toronto and a master's degree in biomedical sciences from Boston University's School of Medicine.

The MITRE Corporation
202 Burlington RD
Bedford, MA 01730-1420
E-mail: ssemy@mitre.org

Steve Foote is a director of engineering for The MITRE Corporation's Command and Control Center, supporting DoD acquisition programs. Most recently, he served as chief architect for the Air Force's Electronic Systems Center, where he facilitated the adoption of net-centric technologies by creating the Net-Centric Momentum Series. Foote has a bachelor's degree in electrical engineering and a master's degree in advanced information systems, both from Northeastern University, and is currently working toward an advanced degree in computer science.

The MITRE Corporation
202 Burlington RD
Bedford, MA 01730-1420
Phone: (781) 266-9521
E-mail: sfoote@mitre.org

Resistance as a Learning Opportunity

David P. Quinn
MOSAIC Technologies Group, Inc.

Many people treat resistance to change as something to overcome. They prepare for battle and arm senior executives with the tools to "beat people into submission." Perhaps what is really happening is that the people introducing the change are not open to it—and are the real source of resistance. This article discusses why resistance exists, its benefits, and how to use it for improvement.

Imagine a small group within your organization coming to you and saying they have the solution to your problem. You don't think that there's a problem, so you are both insulted and highly resistant to the actions you are told to take. The group may try an intervention activity to convince you of the problem. To get rid of them, you just go through the motions of their suggestions and change—but return to old habits as soon as the group stops looking.

The process improvement equivalent to this situation usually occurs when an engineering process group (EPG) approaches software and systems engineering professionals with a set of defined processes that solve their development, maintenance, and management problems. The professionals don't believe they have problems, so they resist the EPG's efforts. The EPG tries, in many ways, to coerce the professionals into compliance. The professionals may decide to go through the motions in order to get rid of the EPG, but return to their old ways once the EPG stops looking.

Another situation is when your EPG does not appear to face resistance. Perhaps you didn't recognize the resistance in your organization because it was disguised in feigned compliance.

Whatever situation you are in, the resistance is natural, understandable, and, surprisingly, desirable. We should not view resistance as something to overcome, but as an opportunity to improve the organization.

Developing Standard Processes

Many organizations have great success working process improvement in accordance with the SEI's CMMI. Central to the CMMI is the principle that the organization should have a standard set of defined processes that their engineering and management professionals use to build products and provide services. Using these standard processes makes teams more productive in a shorter time, and improves the predictability of results.

Organizations expend a great deal of effort and cost to develop a standard set of processes. They form process action teams to define the processes. The EPG provides training on the processes and conducts pilot projects to ensure their usability. Managers and technicians change their normal operating procedures to adapt. They justify all of this expense by citing the anticipated improvements in the product and productivity.

However, some professionals resist these *improvements* in favor of *business as usual*. Organizations take precautions to avoid these instances of resistance through change management principles and enforcement practices. Despite these precautions, there will be resistance to changes in engineering and management processes. The organization's reaction to this resistance indicates how successful it will be with long-term process improvement.

Causes and Levels of Resistance

Resistance to changes in engineering and management processes is a natural reaction because changes challenge a professional's ego. Just suggesting a change insinuates that their current practices are wrong. Since professionals seldom feel that their current practices are wrong, they are unlikely to see a need for change. No matter how positive the change or compelling the argument, the threat to the professional's ego will cause resistance.

However, the level of resistance differs among professionals:

- Those who are open to improvements drop their resistance quickly and adopt the change.
- Cautious professionals drop their resistance and adopt the change when they see that it is an improvement.
- One set of professionals will maintain their resistance no matter how much evidence is given that the change is positive. These resisters make process improvement the enjoyable challenge that it is.

Addressing Resistance Through Coercion

When an EPG encounters resistance, their first reaction is to find some way to overcome it. They address the resistance through a series of intervention sessions. They try to coerce the resisters by talking about their assumed problems. They quote CMMI, invoke the words of senior management, and threaten to impact the resisters' performance review.

The EPG members often turn to coercion, but these attempts are more likely to toughen a resister's stance. The resisters stand firm in their beliefs and may even flaunt their resistance. In order to satisfy the EPG, resisters may go through the motions of the process but are unlikely to retain any change in behavior without constant attention.

Linking Resistance to Problems

An EPG must suppress its own natural reaction to *beat back* resistance. They must consider the resisters' perspective, who likely feel under attack. Maybe they feel like the resisters in *Star Trek: The Next Generation*, who heard the Borg say: "Resistance is futile ... you will be assimilated." The EPG should look for the root cause of the resistance; usually when it is found, the EPG stops appearing to attack and starts resolving problems.

One likely cause of resistance could be that the resisters actually do not suffer the ills the defined process addresses. It makes no sense to provide professionals a software configuration management (SCM) process when they are doing a good job of SCM. The resisters are not in denial; they truly do not have a problem.

Similarly, the resisters may be experiencing the problem the process resolves, but it is not their biggest problem. This may be best described as the Pareto Principle (also known as the 80-20 rule) in reverse. The EPG provides a solution to the 80 percent of the work that causes 20 percent of the problems. For instance, the EPG may provide a defined peer review process that would help the resisters, but does not provide as much help as a requirements elicitation process.

It is also possible that the defined process addresses the right problem but is the wrong solution. For example, I once

was having SCM problems on a project and needed help. Management mandated the use of a particular tool to help resolve everyone's SCM problems. Unfortunately, it was a UNIX tool on a network system while my project was an RSX-11, standalone system. My resistance to the mandated solution was strong and justified. The solution caused as many problems for me as it was supposed to solve.

Obviously, there will be times when the resisters are simply in *total denial* of engineering and management problems. Instead of viewing this as something to overcome, use these resisters for comparisons and possible solutions.

Learning From Resistance

The EPG can transform both the root causes of resistance and total denial into learning opportunities. When the resisters do not have a problem and the solution is not a solution for them, resisters provide the EPG with another addition to the organization's set of standard processes.

When the process addresses the wrong problem, the resistance helps the EPG to prioritize which processes to define and improve next. Resistance due to the wrong solution allows the EPG to adjust its defined process to address a new problem area.

Resisters in denial can also provide a learning opportunity. By using resisters as reviewers, the EPG can learn that certain steps in the process do not add value. The EPG should approach these resisters with questions such as: "How do you handle this situation?" or "What would help you in this situation?" The answers allow the EPG to adjust and improve the defined process, and possibly gain buy-in from the resisters.

Resisters in denial provide the EPG with a performance baseline for comparison when determining whether a defined process is an improvement. If the defined process is an improvement (and this is not always a safe assumption), the resisters should realize they do have a problem and look to adopt the defined process.

Additionally, this baseline comparison allows the organization to determine if the improvement is significant enough to warrant extended use. I'll always remember the time I changed a sort program that shaved three seconds from a five-minute program. It was by definition an improvement, but it did not warrant the effort to develop or implement the change[1]. A baseline comparison lets the organization perform a quantitative cost-benefit analysis for its decision-making process.

Conclusion

When encountering resistance while deploying a defined process, the EPG should not complain or prepare for battle; they should instead work to determine the root cause of resistance. By addressing the root cause, the EPG can learn about another acceptable process, problems in its defined process, or where future improvement efforts need to be focused. The resisters can be used to baseline the defined process' level of improvement. There is even the potential that the resisters will become users.

The EPG should rejoice when it encounters resistance. It has discovered a learning opportunity.◆

Note

1. When you look at shaving three seconds off a 300-second activity, there is not sufficient change to consider it an improvement. A 1 percent change usually does not fall within what an organization considers to be an improvement as specified in the Organizational Innovation and Deployment process area. Organizations will usually set their improvement thresholds around 10 percent before deeming a change to be an improvement and deploying it.

About the Author

David P. Quinn is the managing director for process services at MOSAIC Technologies Group, Inc. He has more than 25 years of software and systems development, maintenance, and management experience. Quinn also has more than 15 years of experience as a process improvement consultant. He is a certified SCAMPI[SM] lead appraiser for CMMI for Development as well as a certified Introduction to the CMMI for Development instructor. He was also a member of the SW-CMM Advisory Board for two years.

MOSAIC Technologies Group, Inc.
8161 Maple Lawn BLVD
STE 430
Fulton, MD 20759
Phone: (717) 451-2149
E-mail: dquinn@mosaicsgroup.com

[SM] SCAMPI is a service mark of Carnegie Mellon University.

Raiders of the Lost Art

It's a brave new world and change seems to be the norm, the motto, and the panacea to its tribulations. Apprehension is in the air; trepidation around the corner. You can feel it lurking, sinister, up to no good.

Is it the financial crisis? No worry there, just print more money. The Influenza Pandemic? Sure, when pigs fly or birds wallow. Climate change? Yes, it always has and always will. No, this is something more subtle; an undercurrent gently sweeping across the societal landscape.

We are slowly losing the art of communication. Don't believe it? Tweet your peeps or text your teenager. See what they say or don't say. The glorious evolution from pictographs to Shakespeare is now declining to digital grunts and tawdry tweets. Face time gives way to Facebook, expression is replaced by emoticons, and the nimble thumb is now the voice of a new generation.

How did this happen? How did we ebb from Kipling to Kardashian? How are millions captivated by Paula Abdul's quest to form a coherent sentence while Huxley's new world remains unexplored?

Do we really care what Lance Armstrong is eating for breakfast and with whom? Do we need to know what Amy Winehouse is rehabbing from this week? We already know what Karl Rove thinks, so why belabor the point? Is Taylor Swift really behind those tweets or is she tweet-synching? Maybe it's a tweet double or Twitter assistant?

Why would anyone want to broadcast their every move? Why would anyone want to stalk someone's every move? "Yea, Britney went number two in the loo." Oh yeah, money. She tweets and you download a song. He blogs and you give to his cancer foundation. Follow the hash-tag to find the money. But in the process, we are losing the art of communication.

"But Gary, from an engineering standpoint, it's more efficient." Is it? Efficiency is a double-edged sword. Sure, one aspect of efficiency is performing with a minimum of wasted time, effort, and resources. However, an equally important aspect of efficiency is performing in the best possible manner. It cuts both ways. If your communication is ineffective, it doesn't matter if it is quick, ubiquitous, or efficient.

Now the bad news: I regret to report that engineers have their fingerprints all over the crime scene. Those in our own profession are the very raiders of the lost art of communication. You don't need a code from Da Vinci to realize that leaving the redesign of social networks in the hands of communications engineers is not the best of ideas. Would you consign to Robin Williams timidity, John Madden flight, or Janeane Garofalo tea parties? No, it's not in their nature. So why hand over social relations to a communications engineer?

Come on, I love engineers. I was one once, but you don't turn over social networking to the prodigies who preferred calculating instantaneous rates of change to cultivating colleagues and companions. Most engineers are lucky if they find a Sancho Panza, let alone pursue the impossible dream of wooing their Dulcinea.

Okay, maybe I'm being too harsh. Engineers are amiable blokes and cordial lasses, but it is a tell-tale sign if you are invited to the party to wire the sound system rather than to be wired. If the only buzz you get is electrical in nature, it should cause you to pause and think.

I know someone has to roll up their sleeves to parse ambiguous requirements, calculate complex algorithms, decipher convoluted code, and exploit volatile technologies. Not everyone can garner fulfillment from extended screen exposure, elusive bugs, fast Fourier transforms, and five lines of über-efficient code. We need you, we value you, please stick around—but realize to communicate is human, not mechanical.

Here are a few suggestions that may help you digitally degauss and socially recharge.

1. Turn off the computer. Leave your cell phone and iPod at home. Invite family or friends on a walk. Talk about anything, but talk the whole walk. If you are not talking, stop walking. When you start talking you can walk.
2. Read a book. Not a technical manual, not a comic book ... and you can't use your Kindle. Preferably classic literature (but not from iPhone's "Classics" app). Don't race to the end; instead, savor the story.
3. Take in a live concert, musical, play, or comedy.
4. Learn the basics of human communication—transmission, reception, as well as verification and validation—without the use of technology.
5. Add someone to your design team who knows absolutely nothing about technology or can recount the salient points of "Les Misérables" in one minute.

The very technologies designed to bring us together are keeping us apart. They promote isolation instead of collaboration. It's like the New Yorker who proudly pronounced to his country cousin, "Everything I need—my office, my apartment, the grocery store, the drug store, and theatre—are all within a city block. I never have to leave." His cousin scratched his head and inquired, "And that's a good thing?"

In designing our brave new world—where everything is a click away—remember to ask yourself if that's a good thing.

Let Britney go and spend some time with Billy, the Bard of Avon. "O wonder! How many goodly creatures are there here! How beauteous mankind is! O brave new world! That has such people in't!"[1]

…not sent via my BlackBerry, my iPhone,
Twitter, or carrier pigeon.
—Gary Petersen, Arrowpoint Solutions, Inc.

Note

1. Shakespeare, William. The Tempest. Act V, Scene I.

Can You BACKTALK?

Here is your chance to make your point without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. These articles should provide a concise, clever, humorous, and insightful perspective on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery, and should not exceed 750 words.

For more information on how to submit your BACKTALK article, go to <www.stsc.hill.af.mil>.

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737

CrossTalk thanks the above organizations for providing their support.